НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Система збору та аналізу метрик сервера»

Виконав:
студент IV курсу, групи ІП-62
Павленко Віталій Миколайович _____

Керівник:
кандидат технічних наук, доцент
Корочкін Олександр Володимирович _____

Консультант з нормокнтролю:
професор, доктор технічних наук
Сімоненко Валерій Павлович _____

Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Павленку Віталію Миколайовичу

1. Тема проєкту «Система збору та аналізу метрик сервера», керівник проєкту Корочкін Олександр Володимирович, кандидат технічних наук, доцент, затверджені наказом по університету від «07» травня 2020 р. № 1081-с

2. Термін подання студентом проєкту _____

3. Вихідні дані до проєкту: теоретичні відомості, технічна документація, система для збору та аналізу метрик сервера.

4. Зміст пояснювальної записки: аналіз предметної області та огляд існуючих систем, вибір інструментів розробки, розробка програмного забезпечення та аналіз розробленого програмного забезпечення.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): схема структури бази даних, структурна схема системи.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Сімоненко В. П., професор		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Затвердження теми роботи	16.12.2019 - 20.12.2019	
2	Вивчення та аналіз завдання	20.12.2019 - 03.02.2020	
3	Розробка архітектури та загальної структури системи	03.02.2020 - 24.02.2020	
4	Розробка структур окремих підсистем	24.02.2020 - 16.03.2020	
5	Програмна реалізація системи	16.03.2020 - 11.05.2020	
6	Виправлення помилок	11.05.2020 - 01.06.2020	
7	Оформлення пояснювальної записки	09.06.2020	
8	Передзахист	20.06.2020	
9	Захист	16.12.2020 - 20.12.2020	

Студент Віталій ПАВЛЕНКО

Керівник Олександр КОРОЧКІН

АНОТАЦІЯ

У даній роботі було детально розглянуто існуючі системи для збору та аналізу метрик сервера, а також їх принцип роботи. Були проаналізовані їхні слабкі та сильні сторони, їх принципи роботи. На основі існуючих рішень було запропоновано створити нову систему, яка б вирішувала основні мінуси існуючих систем. Система збору та аналізу метрик сервера була розділена на клієнтську та серверну частини, які взаємодіють між собою. В результаті роботи була розроблена система збору та аналізу метрик сервера, що не гірша за існуючі аналоги та вирішує їхні недоліки.

ANNOTATION

This work discusses in detail the existing systems for collecting and analyzing server metrics, as well as their principle of operation. Their weaknesses and strengths, their principles of work were analyzed. Based on existing solutions, it was proposed to create a new system that would fix the main disadvantages of existing systems. The system of collecting and analyzing server metrics was divided into client and server parts, which interact with each other. As a result, a system for collecting and analyzing server metrics was developed, which is not worse than existing analogues and fixes their cons.

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1.	А4		Завдання на дипломний проєкт	2	
2.	А4	ІАЛЦ.467200.001 ВП	Відомість на дипломний проєкт	1	
3.	А4	ІАЛЦ.467200.002 ТЗ	Технічне завдання	4	
4.	А4	ІАЛЦ.467200.003 ПЗ	Пояснювальна записка	56	
5.	А1	ІАЛЦ.467200.004 Д1	Схема структурна	1	
6.	А1	ІАЛЦ.467200.005 Д2	Діаграма варіантів використання системи	1	
7.	А1	ІАЛЦ.467200.006 Д3	Алгоритм роботи системи	1	
8.	А1	ІАЛЦ.467200.007 Д4	Лістинг програми	39	

				ІАЛЦ.467200.001 ВП		
	ПІБ	Підп.	Дата			
Розробн.	Павленко В.М.			Відомість дипломного проєкту	Лист	Листів
Керівн.	Корочкін О.В.				1	5
					КПІ ім. Ігоря Сікорського Каф. ОТ Гр. ІП-62	
Н/контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЕКТУ

на тему: «Система збору та аналізу метрик сервера»

Київ – 2020

Зміст

					ІАЛЦ.467200.002 ТЗ		
Зм.	Арк.	№ докум.	Підпис	Дата			
Розробив		Павленко В. М.			Система збору та аналізу метрик сервера	Літ.	Аркуш 1 / Аркушів 3
Перевірив		Корочкін О. В.					
Реценз.					Технічне завдання	НТУУ КПІ, ФІОТ, ІП-62	
Н. Контр.		Сімоненко В.					
Затвердив		Стіренко С. Г.					

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання стосується розробки системи збору та аналізу метрик сервера, яка відповідає за збір даних з певних серверів та їх аналіз.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврської дипломної роботи, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут» імені Ігоря Сікорського.

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення простої на гнучкої системи збору та аналізу метрик сервера.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелами розробки є науково-технічна література, монографії, публікації в періодичних виданнях та мережі Інтернет.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Система, що розробляється, повинна:

1) бути розподіленим веб-сервісом, що складається з клієнтської частини, яка відображає дані, сервера який отримує дані та аналізує їх, та консольної утиліти, яка надсилає дані. бути легко

2) розширюваною, не залежною від баз даних, має надавати більший функціонал ніж існуючі рішення.
3) надавати можливість збирати дані сервера, аналізувати їх відносно побажань клієнта, при тому бути простою та зручною у керуванні системою.

5.2. Вимоги до інструментального програмного забезпечення

- середовище з встановленою платформою Node.js версії 14.03 та MongoDB версії 4.0.4;

5.3. Вимоги до апаратної частини обчислювальної системи

- процесор з тактовою частотою не менше ніж 1000 КГц;
- оперативна пам’ять об’ємом не менше ніж 800 Мб;
- жорсткий диск об’ємом не менше ніж 300 Мб.

6 ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури	20.12.2019
Складання і узгодження технічного завдання	03.02.2020
Розробка архітектури та загальної структури системи	16.03.2020
Програмна реалізація системи	11.05.2020
Допрацювання і налагодження програми	01.06.2020
Оформлення документації дипломної роботи	09.06.2020

Пояснювальна записка
до дипломного проєкту
на тему: «Система збору та аналізу метрик сервера»

Київ – 2020 року

Зміст

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БД – База даних

СКБД – Система Керування Базами Даних

CRUD – Create Read Update Delete

ІС – Інформаційна Система

CLI – Command Line Interface

API – Application Programming Interface

SPA – Single page application

DRY – Don’t repeat yourself

YAGNI – You aren’t gonna need it

KISS – Keep it short and simple

ВСТУП

Сучасний світ неможливо уявити без мережі Інтернет. Його неможливо уявити без використання інформаційних технологій. Число користувачів глобальної мережі в світі, росте з кожним днем. Більшість послуг сьогодні надаються через Інтернет, а саме: покупка товарів, перегляд фільмів, обмін даними, спілкування, прослуховування музики та навіть пошук друзів. Зараз майже неможливо представити бізнес без веб-сайта, де б можна було ділитися своїми досягненнями, інформацією про сферу, вплив, директорів та власників, інформацією для інвесторів. Саме тому наявність веб-сайту фірми є скоріше необхідністю, ніж розкішшю.

Актуальність даної теми зумовлена тим, що велика кількість компаній орієнтована на збільшення кількості користувачів та покращення вражень користувачів від їхнього сервісу. Тому постає питання у ефективності веб-додатків, у швидкості роботи застосунку та швидкому виявленні та вирішенні проблем, аби не втратити жодного клієнта та вони не залишали поганих рецензій сервісу.

Розуміння стану інфраструктури та систем важливо для стабільної роботи сервісів. Інформація про працездатність і продуктивності не тільки допомагає команді вчасно реагувати на проблеми, але і дає їй можливість вчасно вносити всі необхідні зміни. Один з кращих способів отримати цю інформацію - це надійна система моніторингу, яка збирає метрики, візуалізує дані і попереджає операторів, коли щось не працює належним чином.

Метрики, моніторинг і система оповіщень - це взаємопов'язані поняття, які разом складають основу системи моніторингу. Вони мають можливість відобразити стан систем, допомогти вам зрозуміти тенденції в споживанні ресурсів або поведінці і зрозуміти вплив змін, які ви вносите. Якщо показники перевищують очікувані діапазони, ці системи можуть відправляти

повідомлення, щоб допомогти оператору своєчасно визначити можливі причини збоїв.

Метрики - це неформатовані дані про використання ресурсів або поведінці, які можна відстежувати і збирати в системах. Це можуть бути звіти про використання, що надаються операційною системою, або дані більш високого рівня, прив'язані до конкретних функцій або компонентів (наприклад, кількість запитів в секунду, членство в пулі веб-серверів і т.п.). Деякі метрики представлені в відношенні до загальної потужності, а інші представлені рейтингами завантаженості компонента. Оскільки ці дані залежать від часу коли їх було зібрано - важливим моментом є їхнє збереження. Адже для подальшого аналізу зібраних даних - важливо розуміти у який час ці дані було отримано.

Часто починати рекомендується з простих метрик - тих, які вже виставлені вашою операційною системою для подання використання основних фізичних ресурсів. Дані про дисковий простір, завантаження процесора, використання свопу і т. д. Вже доступні в системі, миттєво надають значення і можуть бути перенаправлені в систему моніторингу без додаткової роботи.

Для збору інших метрик - особливо це стосується ваших власних додатків - вам може знадобитися додати код або інтерфейси.

Метрики корисні, тому що вони дають уявлення про поведінку і працездатності системи, особливо при загальному аналізі. По суті, метрики - це необроблений матеріал, який система моніторингу використовує для побудови цілісного уявлення про середовище, автоматизації реакції на зміни і відправки попереджень. Метрики - це основні значення, що використовуються для розуміння тенденцій, кореляції різних факторів і відстеження змін в продуктивності, споживанні ресурсів або частоті збоїв.

Це допомагає мережевим адміністраторам та веб-розробникам швидше вирішувати проблеми.

Збір метрик дозволяє отримати більше інформації про споживання ресурсів інфраструктурою. Візуалізуючи зібрані дані про використання ресурсів, можна отримати уявлення про продуктивність системи, закономірності та тренди споживання ресурсів. Система сповіщення своєчасно сповістить вас, якщо використання ресурсів вийде за допустимі межі.

Тому тема даного дослідження є актуальною, а отримані результати мають теоретичну та практичну цінність.

РОЗДІЛ 1

ОГЛЯД ІСНУЮЧИХ СИСТЕМ МОНІТОРИНГУ ВЕБ-СТОРІНОК

1.1. Опис предметного середовища

Поняття «веб-сервер» може ставитися як до апаратної начинки, так і до програмного забезпечення. Або навіть до обох частин, які працюють спільно.

З точки зору "заліза", веб-сервер - це комп'ютер, який зберігає файли сайту (HTML-документи, CSS-стилі, JavaScript-файли, картинки та інше) і доставляє їх на пристрій кінцевого користувача (веб-браузер і т . Д.). Він підключений до мережі Інтернет і може бути доступний через доменне ім'я, подібне kpi.ua.

З точки зору програмного забезпечення, веб-сервер включає в себе кілька компонентів, які контролюють доступ веб-користувачів до розміщених на сервері файлів, як мінімум - це HTTP-сервер. HTTP-сервер - це частина програмного забезпечення, яка розуміє URL'и (веб-адреси) і HTTP (протокол, який ваш браузер використовує для перегляду веб-сторінок).

На самому базовому рівні, коли браузеру потрібен файл, розміщений на веб-сервері, браузер запитує його через HTTP-протокол. Коли запит досягає потрібного веб-сервера ("залізо"), сервер HTTP (ПО) приймає запит, знаходить запитуваний документ (якщо немає, то повідомляє про помилку 404) і відправляє назад, також через HTTP [7].

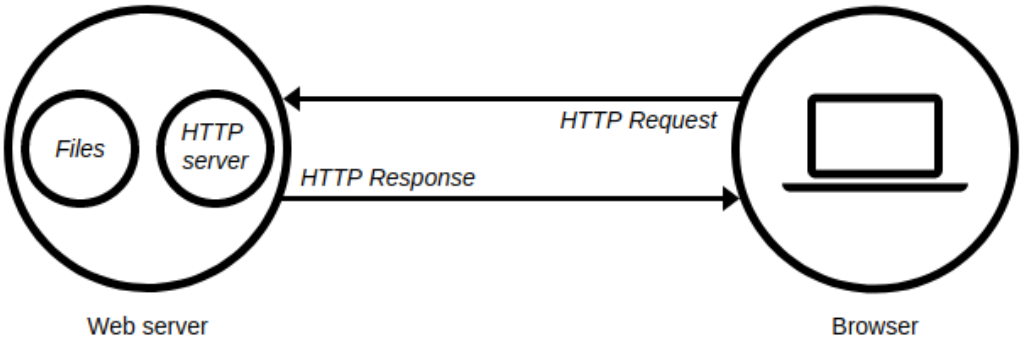


Рис. 1.1 Схема взаємодії веб-сервера з користувачем

Щоб опублікувати веб-сайт, необхідний або статичний, або динамічний веб-сервер.

Статичний веб-сервер, або стек, складається з комп'ютера ("залізо") з сервером HTTP (ПО). Ми називаємо це «статикою», тому що сервер посилає розміщені файли в браузер «як є».

Динамічний веб-сервер складається з статичного веб-сервера і додаткового програмного забезпечення, найчастіше сервера додатки і бази даних. Ми називаємо його «динамічним», тому що сервер додатків змінює вихідні файли перед відправкою в ваш браузер по HTTP.

Наприклад, для отримання підсумкової сторінки, яку ви переглядаєте в браузері, сервер додатків може заповнити HTML-шаблон даними з бази даних. Такі сайти, як MDN або Вікіпедія, складаються з тисяч веб-сторінок, але вони не є реальними HTML документами - лише кілька HTML-шаблонів і гігантські бази даних. Ця структура спрощує і прискорює супровід веб-додатків і доставку контента.

Щоб завантажити веб-сторінку, як ми вже говорили, ваш браузер відправляє запит до веб-сервера, який приступає до пошуку запитуваної файлу в своєму власному просторі пам'яті. Знайшовши файл, сервер зчитує його, обробляє як йому це необхідно, і відсилає в браузер. Давайте розглянемо ці кроки більш докладно.

Перш за все, веб-сервер повинен містити файли веб-сайту, а саме: HTML-документи і пов'язані з ними ресурси, включаючи зображення, CSS-стилі [8], JavaScript-файли, шрифти ...

Технічно, ви можете розмістити всі ці файли на своєму комп'ютері, але набагато зручніше зберігати їх на виділеному веб-сервері, який:

- завжди запущений і працює
- завжди підключений до Інтернету

- має незмінний IP адреса (не всі провайдери надають статичний IP-адресу для домашнього підключення)
- обслуговується третьої, сторонньою компанією

Подруге, веб-сервер забезпечує підтримку HTTP (англ. *Hypertext Transfer Protocol* - гіпертекстовий транспортний протокол). Як випливає з назви, HTTP вказує, як передавати гіпертекст (тобто пов'язані веб-документи) між двома комп'ютерами.

Протокол являє собою набір правил для зв'язку між двома комп'ютерами. HTTP є текстовим протоколом без збереження стану.

Текстовий

Всі команди є простим людино читаємим текстом.

Не зберігає стан

Ні клієнт, ні сервер не пам'ятають про попередні з'єднаннях. Наприклад, спираючись тільки на HTTP, сервер не зможе згадати введений пароль або на якому етапі транзакції ви перебуваєте. Для таких завдань, вам буде потрібно веб-додатки.

HTTP задає суворі правила взаємодії клієнта і сервера:

- Виключно клієнти можуть виробляти HTTP-запити, і тільки на сервер. Сервера здатні тільки відповідати на HTTP-запити клієнта.
- При запиті файлу по HTTP, клієнт повинен сформувати файловий URL.
- Веб-сервер повинен відповісти на кожен HTTP-запит, по крайній мірі - повідомленням про помилку.

На веб-сервері HTTP-сервер відповідає за обробку вхідних запитів і відповідь на них.

1. При отриманні запиту, HTTP-сервер спочатку перевіряє, чи існує ресурс по данному URL.
2. Якщо це так - веб-сервер відправляє вміст файлу назад в браузер. Якщо немає, веб-додаток генерує необхідний ресурс.

3. Якщо ніщо з цього не можливо, веб-сервер повертає повідомлення про помилку клієнту, найчастіше "404 Not Found". (Це помилка настільки поширена, що багато веб-дизайнерів витрачають велику кількість часу на розробку 404 сторінки про помилку.)

Грубо кажучи, сервер може віддавати статичний або динамічний вміст. «Статичний» означає «віддається як є». Статичні веб-сайти - найпростіші веб-сайти.

«Динамічний» означає, що сервер обробляє дані або навіть генерує їх на льоту з використанням бази даних. Це забезпечує більшу гнучкість, але технічно складніше в реалізації і обслуговуванні, через що процес створення сайту дуже сильно ускладнюється.

Існує так багато серверів додатків, що досить важко запропонувати якийсь один. Деякі сервери додатків заточені під певні категорії веб-сайтів, такі як блоги, вікі-сторінки або інтернет-магазини; інші, звані CMSs (системи управління контентом), більш універсальні.

На сьогодні можна знайти багато готових рішень для написання веб-серверів - Spring, .Net, Express, Zio, Django, тощо. Вони прискорюють розробку та запобігають виникненню великої кількості помилок.

1.2. **Короткі теоретичні відомості**

Метрики - це неформатовані дані про використання ресурсів або поведінці, які можна відстежувати і збирати в системах. Це можуть бути звіти про використання, що надаються операційною системою, або дані більш високого рівня, прив'язані до конкретних функцій або компонентів (наприклад, кількість запитів в секунду, членство в пулі веб-серверів і т.п.) [1].

Моніторинг – це процес безперервного збору даних за певними індикаторами, що забезпечує зацікавлених осіб показниками досягнення цілей системи. Тобто визначення можливості отримати доступ до ресурсів

сайту зі сторони випадкового клієнта. Якщо ж відвідувач не може завантажити потрібну сторінку, знайти інформацію, увійти до особистого кабінету чи довго чекає завантаження контенту сайту, то даний веб-сайт можна вважати неефективним. Він не виконує головного завдання – задоволення потреб користувача. Саме для простежування ефективності роботи веб-сайту використовують моніторинг.

Продуктивність - це загальна міра ефективності системи. Це широкий термін, який часто охоплює такі фактори, як пропускна здатність, латентність або споживання ресурсів.

Візуалізація - це процес презентації метрик в візуально зручному форматі (графіку, діаграми і т.п.), що дозволяє швидко і інтуїтивно інтерпретувати дані.

Інструментація - це здатність відстежувати поведінку і продуктивність програмного забезпечення. Вона досягається шляхом додавання в програмне забезпечення коду і конфігурації для виведення даних, які потім можуть бути використані системою моніторингу.

Ефект спостерігача - це вплив самої системи моніторингу на спостережувані метрики. Оскільки моніторинг використовує ресурси, вплив системи моніторингу буде впливати на продуктивність і набутих значень. Системи моніторингу прагнуть уникати збору непотрібних даних для мінімізації цього впливу.

Поріг - це межа між допустимими і неприйнятними значеннями, які викликають попередження. Часто попередження спрацьовують, коли метрика перевищує порогове значення протягом певного періоду часу (щоб уникнути відправки попереджень про тимчасові сплески трафіку).

HTTP - протокол для передачі даних, належить до 7-го рівня моделі OSI, найпопулярніший протокол на сьогодні у комп'ютерних мережах. Назва – це скорочення від **H**yper **T**ext **T**ransfer **P**rotocol, тобто протокол для передачі гіпер-текстових файлів.

1.3. Огляд наявних аналогів

Icinga [2] (рис. 1.2) - безкоштовний сервіс для збору метрик, моніторингу та аналізу зібраних даних. Проте для його роботи потрібен власний сервер, на якому буде запущений сервіс, який збиратиме дані з серверів. Якщо використовувати Icinga як сервіс, то за це доведеться платити. Якщо ви хочете мати особисту підтримку та допомогу з цим сервісом - доведеться платити чималу суму (від 1000$).

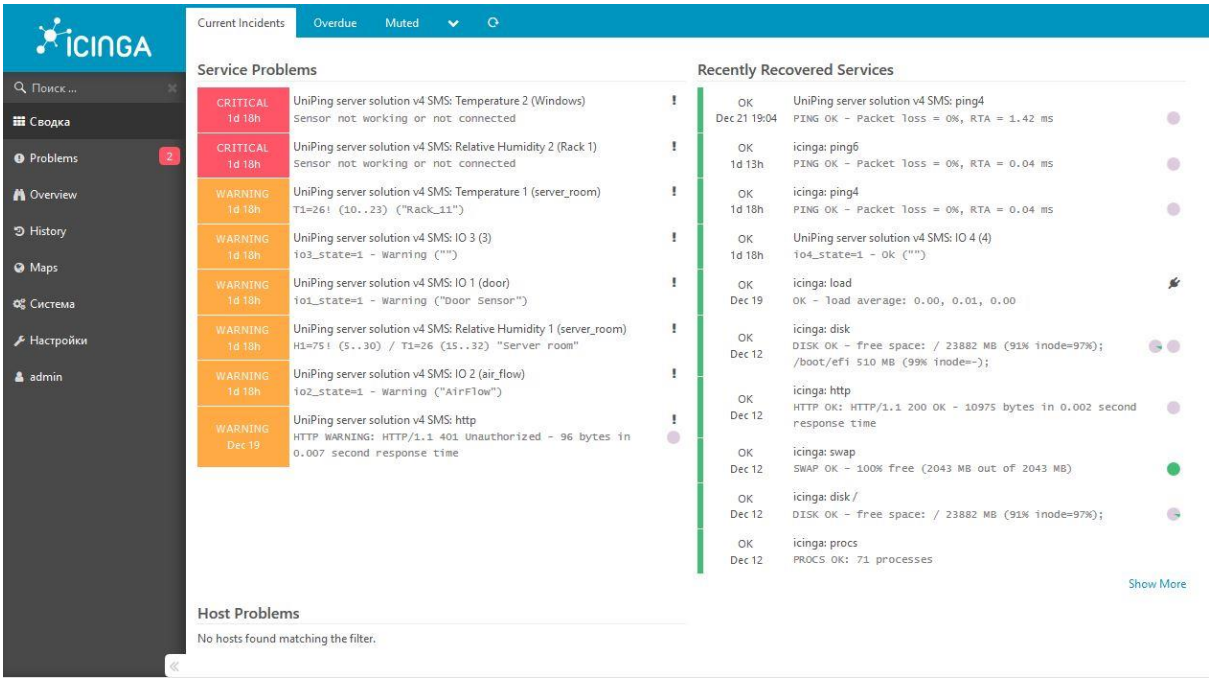


Рис. 1.2 Інтерфейс сервісу Icinga

Перечень можливостей:

- Моніторинг служб мережі (IMAP, POP3, SMTP, HTTP, Ping і т. д.)
- Моніторинг ресурсів хоста (завантаження ЦП, використання дисків, використання оперативної пам'яті)
- Моніторинг серверних компонентів (комутатори, маршрутизатори, сервери, датчики температури, вологості і т. Д.)
- Можливість створення власних плагінів (через нагромаджений функціонал створення плагінів не досить зручно, але можливе)

- Можливість автоматичної відправки повідомлень по E-Mail

Візуальне оформлення та звіти:

- Інтерфейс Icinga Web 2 для відображення статусу служб і пристроїв
- База звітів з різним рівнем доступу і автоматичним створенням звітів
- Звіти про використання потужностей
- Графіки стану і продуктивності (доступно через плагіни)

Nagios [3] - промисловий стандарт в моніторингу IT-інфраструктури - open source рішення, призначене для моніторингу комп'ютерних систем і мереж: спостереження, контролю стану обчислювальних вузлів і служб та оповіщення системних адміністраторів у разі припинення або відновлення роботи будь-яких служб. Крім безкоштовної версії Nagios Core, існує комерційна Nagios XI з додатковими можливостями, що володіє більш сучасним і простим в навігації web-інтерфейсом, який пропонує інтерактивну інформаційну панель з оглядом хостів, сервісів і мережевих пристроїв. На його основі та прикладі був створений сервіс Icinga.

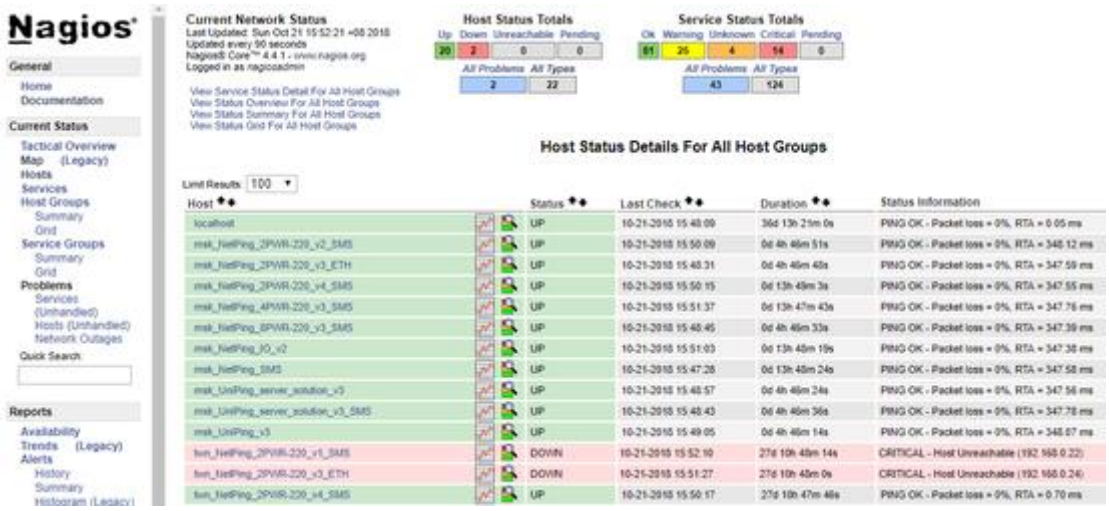


Рис. 1.3 Інтерфейс сервісу Nagios

Перелік можливостей:

- Протоколи моніторингу мереж та їхніх служб (SMTP, POP3, HTTP, NNTP, Ping і т. д.)

- Моніторинг ресурсів хоста (завантаження ЦП, використання дисків, використання оперативної пам'яті)
- Моніторинг серверних компонентів (комутатори, маршрутизатори, сервери, датчики температури, вологості і т. д.)
- Можливість створення власних плагінів (простіше чим у Icinga)
- Можливість автоматичної відправки повідомлень по E-Mail
- Автоматична ротація лог-файлів
- Параллельна перевірка служб
- Підтримка спостерігання через шифровані тунелі, тобто SSH та SSL

Основними недоліками є:

- Немає можливості конфігурації через web-інтерфейс (для безкоштовної версії). Всі зміни конфігурації виконуються правкою файлів конфігурації з подальшим повним перезапуском сервера Nagios (~ 10-15 хвилин).
- Занадто великий інтервал між перевірками і вимірами параметрів.
- Відсутні вбудовані засоби візуалізації (лише через віджети).
- Кожен плагін запускається як окремий процес.
- Усереднює дані, тому неможливо сказати, яке було точне значення параметрів, наприклад, місяць тому.

Перевагами є:

- Розвинута підтримка користувачів - існують плагіни на всі випадки життя від сторонніх виробників.
- Дозволяє залишати коментарі з міткою часу.
- Простий формат конфігураційного файлу.

Zabbix [4] - це безкоштовна система моніторингу для відстеження статусів різноманітних сервісів комп'ютерної мережі, серверів та мережевого обладнання. Zabbix є open source enterprise рішенням, яке може виробляти комплексний моніторинг інфраструктури (сервери, мережеві пристрої і віртуальні машини), візуалізувати отриману інформацію в графіки, стежити

за навантаженням і продуктивністю обладнання з використанням власних агентів, що підтримуються практично всіма операційними системами).

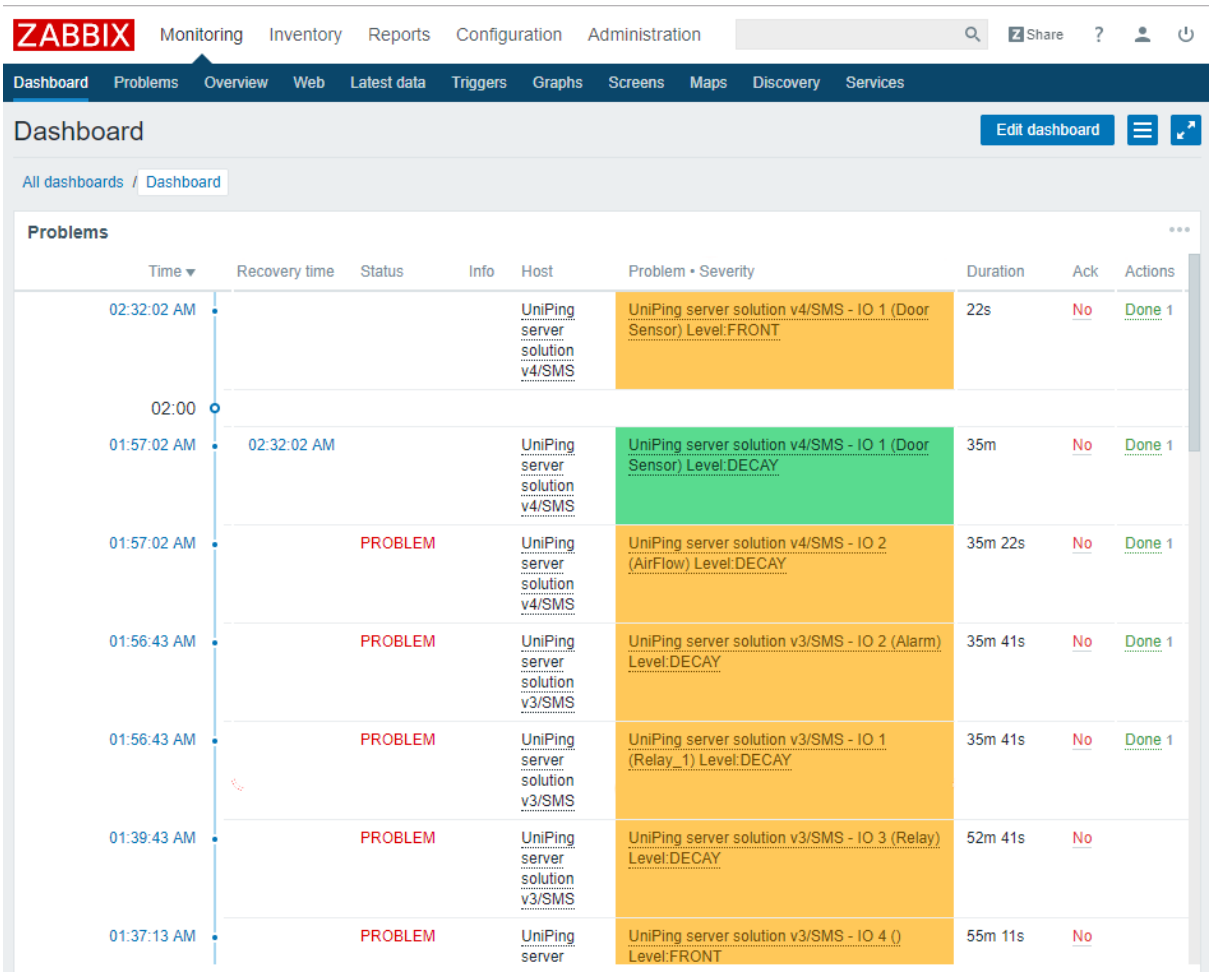


Рис. 1.4 Інтерфейс сервісу Zabbix

Основними недоліками є:

- Виконується через постійно запущений окремий процесс.
- Всі дані моніторингу зберігаються в базі, що в великих мережах вимагає виділення додаткових обчислювальних потужностей для обслуговування бази даних.

Перевагами є:

- Доступність (сервіс є абсолютно безкоштовним);
- Конфігурація через web-інтерфейс і за допомогою API.
- Єдина точка доступу для користувачів.

- Розмежування доступу до даних і конфігурації.
- Мінімальний інтервал між вимірами - 1 секунда.
- Час зберігання даних обмежена лише дисковим простором.
- Розвинені можливості аналізу зібраних даних.

Prometheus [5] - безкоштовна система моніторингу, що володіє можливостями тонкої настройки метрик. Вона буде корисна для відстеження стану роботи сервісів на низькому рівні. Система збирає дані за допомогою HTTP запитів. Насправді Prometheus це система компонентів:

- Prometheus Server - відповідає за збір на збереження інформації, також має простий графічний інтерфейс для перегляду даних.
- База даних - місце збереження зібраних метрик.
- Alert Manager - відповідає за надсилання сповіщень адміністратору
- Node Exporter - сервіс для експорту та імпорту даних.

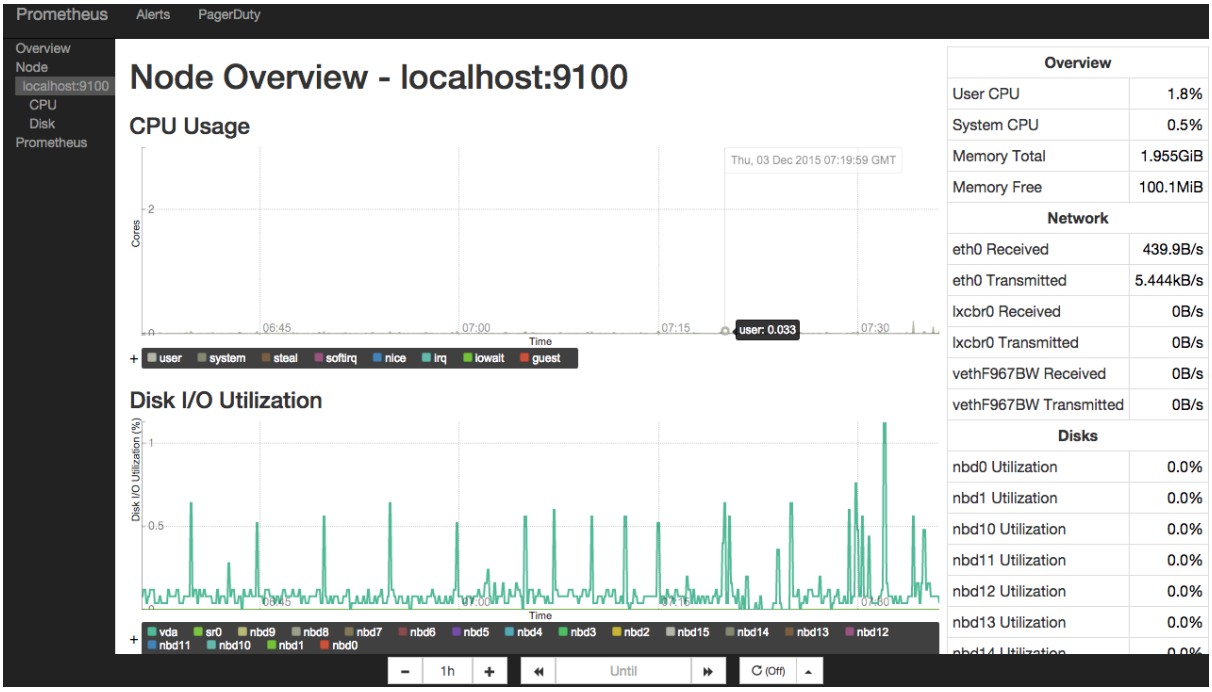


Рис. 1.5 Інтерфейс сервісу Prometheus

Основними недоліками є:

- Занадто простий веб-інтерфейс, тому доводиться використовувати сторонні сервіси, наприклад Grafana.

- Всі дані моніторингу зберігаються в базі, що в великих мережах вимагає виділення додаткових обчислювальних потужностей для обслуговування бази даних.
- Відсутня підтримки збирання логів сервера

Перевагами є:

- Дуже гнучкий у налаштуванні.
- Сучасний, має велику підтримку серед користувачів та використовує модерні рішення.
- Має модульну структуру - що дозволяє використовувати потрібні лише вам сервіси.
- Не конектиться до фізичного сервера напряму, а збирає дані через HTTP.
- Сервіс є абсолютно безкоштовним.

Netdata [6] - це проста утиліта для моніторингу сервера. Вона укомплектована великою кількістю плагінів (Mysql, Nginx і купою інших) і прекрасною системою візуалізації.

Рис. 1.6 Інтерфейс сервісу Netdata

Основними недоліками є:

- Збираються й аналізуються лише локальні дані, тобто неможливо дивитися дані одразу з декількох сервісів.

- Неможливо дізнатися якщо сервер не працює.

Перевагами є:

- Оскільки дані знаходяться та аналізуються локально - можливо вести спостереження за сервісом у реальному часі.
- Дуже простий у налаштуванні.
- Має HTTP інтерфейс, тобто дані зібрані Netdata можуть потім збирати інші системи.
- Велика кількість плагінів.
- Сервіс є абсолютно безкоштовним.

1.4. Порівняльний аналіз наявних програмних продуктів

Більш доцільно буде показати результати порівняння у вигляді таблиці (таблиця 1.1), що містить основні відмінності наявних програмних продуктів.

Оскільки функціонал всіх сервісів майже однаковий, то буде наведена основна відмінність між ними. Також всі сервіси мають безкоштовну версію, тому порівняння по ціні відсутнє.

Таблиця 1.1 Порівняння наявних програмних продуктів

Назва сервісу	Рівень навичок у використанні	Підтримка	Платформа	Основні недоліки	Основні відмінності
1	2	3	4	5	6
Icinga	Потрібні базові навички	Online Платні: Business, Remote Debugging	Cloud, SaaS, Web	Окремий процес на сервері для збирання метрик, недостатньо простий у налаштуванні	Можливість створення значних звітів та зручна конфігурація їх налаштувань

Таблиця 1.1(закінчення)

1	2	3	4	5	6
Nagios	Потрібні базові навички	Online Платні: Business	Cloud, SaaS, Web	Окремий процес для збирання метрик, недостатньо можливостей у налаштуванні	Зручний для написання власних плагінів та має велику кількість плагінів від користувачів
Zabbix	Потрібні базові навички	Online Платні: Business	Cloud, SaaS, Web	Окремий процесс на сервері для збирання метрик	Розвинені можливості аналізу зібраних даних, можливість гнучкого доступу до даних
Prome-theus	Особливі навички не потрібні	-	Web	Занадто простий веб-інтерфейс	Гнучкий у налаштуванні, модульна структура, збір даних по HTTP.
Netda-ta	Особливі навички не потрібні	-	Web	Працює локально	Моніторинг в реальному часі

Висновки до розділу 1

Проведений огляд наявних сучасних засобів збору метрик, показав, що:

1. Основним завданням засобів збору метрик є доступність даних для користувачів, запобігання і оповіщення інформації про проблеми для їх швидкого усунення.
2. Велика кількість веб-сервісів сприяла до автоматизації процесів збору метрик серверів та виявлення проблем з ними, перш ніж вони призведуть до будь-якої серйозної шкоди, що вплине на продуктивність інфраструктури та на остаточних клієнтів.
3. Проаналізувавши готові реалізації інструментів збору метрик було виявлено такі недоліки:
   - Невелика кількість можливостей є безкоштовною;
   - Обмежений стандартний інтерфейс;
   - Складність у налаштуванні;
4. Дивлячись на згадані вище недоліки, можна визначити основні функції системи моніторингу:
   - Легкість у використанні;
   - Доступність усіх можливостей безкоштовно;
   - Зручне у використанні API;
   - Приємний та зрозумілий інтерфейс;

РОЗДІЛ 2

ПРОЕКТУВАННЯ СИСТЕМИ ЗБОРУ ТА АНАЛІЗУ МЕТРИК СЕРВЕРА

2.1. Як працює моніторинг сайту, збір метрик та їх аналіз

Для автоматичного моніторингу веб-сайту використовується мережа комп'ютерів, розташованих поруч з кінцевими користувачами сайту. Ця мережа комп'ютерних контрольних точок взаємодіє з веб-сайтом або службою, щоб переконатися, що служба працює належним чином. Система моніторингу призначає контрольну точку для перевірки майданчика, і контрольна точка може пройти кілька етапів для проведення перевірки[13]. Контрольно-пропускний пункт:

1. Ініціює з'єднання з сайтом або сервісом
2. Перевіряє повернення коду відповіді. Для базової доступності контрольна точка повідомляє про результат і вважає тест завершеним, але для більш розширеного моніторингу контрольна точка триває.
3. Перевіряє повернення зазначеного вмісту
4. Завантажує контент в справжній браузер (Real Browser Monitoring)
5. Записує час завантаження для кожного елемента сторінки при його завантаженні в браузері (моніторинг продуктивності)
6. Спроби увійти в систему, провести пошук, використовувати кошик, навіть зробити покупку (моніторинг веб-додатки)
7. Повідомляє про свої висновки службі моніторингу

Якщо результат включає в себе помилки або повільний час відгуку, служба може почати перевірку знову з іншої контрольної точки, щоб перевірити постійну помилку, перш ніж повідомити службу підтримки сайту.

Моніторинг сайту може здійснюватися як усередині, так і зовні корпоративного брандмауера. Традиційні рішення по управлінню мережею

зосереджені на моніторингу брандмауера, тоді як зовнішній моніторинг продуктивності буде тестувати і відслідковувати проблеми з продуктивністю через магістраль Інтернету, а в деяких випадках аж до кінцевого користувача. Сторонні рішення для моніторингу продуктивності веб-сайтів можуть відстежувати внутрішні (за брандмауером), зовнішні (орієнтовані на клієнта) або хмарні веб-додатки.

Зовнішній моніторинг дозволяє контролювати всі параметри вашого веб-ресурсу. Як приклад:

- Доступність веб-сторінки або швидкість роботи (моніторинг доступності)
- Середній час завантаження веб-сторінки (повний моніторинг сторінки)
- Функціональність веб-сторінки (моніторинг синтетичних транзакцій)
- Стресостійкість веб-сторінки (Web Stress Tester)
- Моніторинг реальних користувачів (кількість користувачів)
- API моніторинг (JMeter)

Внутрішній моніторинг може використовуватися у вашій локальній мережі, коли ресурс, який необхідно відстежувати, недоступний з глобальної мережі. І саме за допомогою спеціальних апаратних пристроїв, які можуть допомогти вам визначити, чи викликана повільна продуктивність ваших внутрішніх додатків: проектуванням додатків, внутрішньою інфраструктурою, внутрішніми програмами або підключеннями до загальнодоступного Інтернету.

У підсумку, внутрішній моніторинг має все, що можливо для зовнішнього моніторингу плюс продуктивність ваших локальних пристроїв, наприклад, локальна мережа, завантаження ЦП сервера, використання пам'яті і дисків, пропускна здатність, споживана вашими мережевими інтерфейсами, різні системні процеси, стан служб, запущених на вашому сервері, або ваші системні події.

2.2. Типи моніторингу сайта

Моніторинг веб-сайтів включає в себе тестування веб-сайтів на доступність, продуктивність і функціональність, а також оповіщення співробітників служби підтримки, коли сторінка працює не так, як очікувалося. Як правило, тип монітора потрапляє в одну з раніше згаданих категорій, хоча більш просунуті монітори можуть охоплювати всі три.

- Моніторинг доступності

Доступність - це час безвідмовної роботи або, іншими словами, забезпечення того, щоб веб-сайт або служба завжди були доступні і в якійсь мірі працювали. Доступність може включати веб-сервіси, домени і сторінки. Моніторинг доступності має наступні ключові функції:

- Почніть менш ніж за 5 хвилин

Легко створюйте поодинокі тести для моніторингу продуктивності і доступності вашого веб-додатки протягом кількох хвилин.

- Максимальний час безвідмовної роботи і задоволеність користувачів

Контролюйте час безвідмовної роботи і час відгуку ваших додатків з декількох географічних точок. Запустіть синтетичні тести, щоб виміряти продуктивність завантаження сайту і викликів API. Моніторинг сценаріїв Puppeteer, які ми використовуємо для імітації потоків користувачів з різних місць.

- Бути ініціативним

Отримувати повідомлення, щоб попередити вас про проблеми, перш ніж вони впливають на користувачів. Ми можемо використовувати інтегровану службу оповіщення для створення політик оповіщення, які зменшують шум оповіщення.

- Точно і швидко визначте причини невдачі

Аналіз результатів тестів допомагає нам визначити точний крок, коли стався збій, і причину збою; наприклад, непрацюючі посилання, зображення великого розміру, повільний пошук або зовнішні запити. Знімки екрану створюються автоматично, щоб допомогти нам діагностувати збої браузера і історичні проблеми з продуктивністю. Завантажуйте звіти про середньомісячну, щотижневої і щоденної доступності та середньому часу відповіді для ваших тестів.

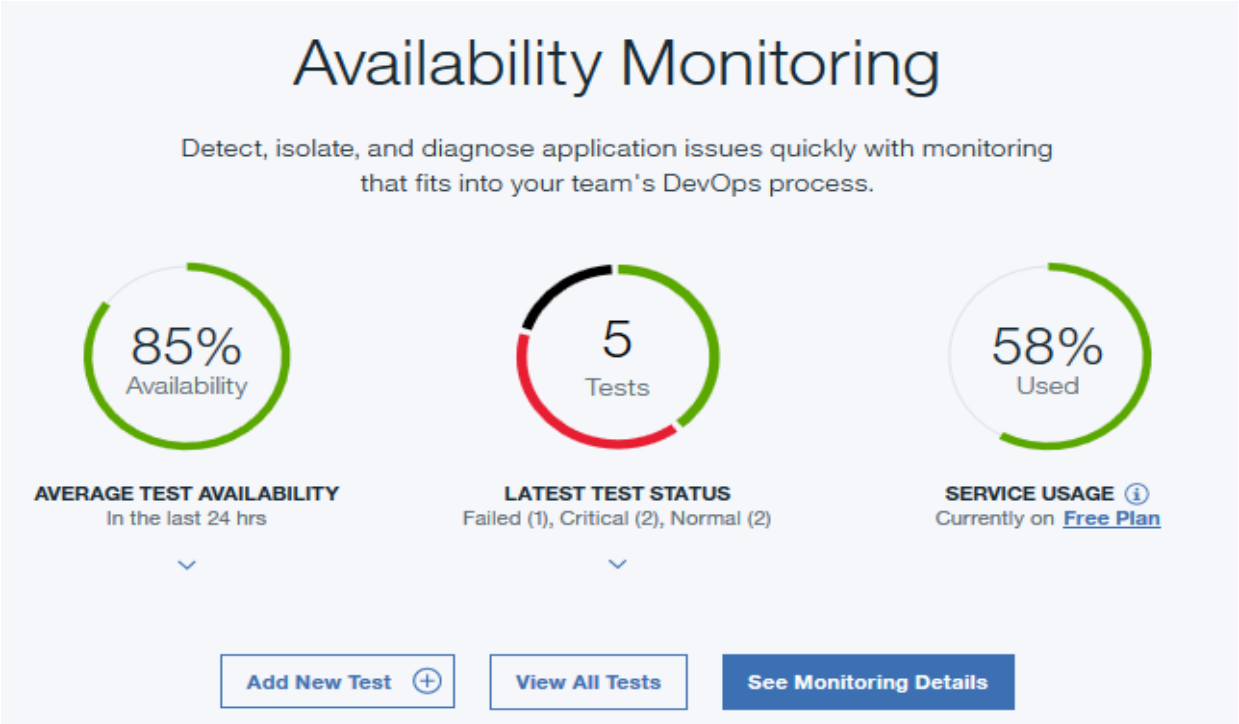


Рис.2.1 Тест моніторингу доступності

- Базовий моніторинг веб-сайтів і API.

Ці базові монітори перевіряють успішний відповідь або конкретну відповідь веб-сайтів і API-інтерфейсів, що підтримують протокол HTTP, і можуть виконувати базову аутентифікацію. Базові монітори доступності також можуть вимірювати час і розмір відповіді і видавати попередження про повільне часу відповіді. Крім цього, існують і такі причини вибору API-моніторингу:

1. Багаторівневий моніторинг

Ми можемо протестувати свої API зверху вниз, налаштувавши кілька HTTP-запитів. Отримувати дані з кожного запиту і використовувати їх для виконання завдань на інших етапах.

2. Монітор для продуктивності

Перевіряти продуктивність кожного запиту і встановлювати затвердження для максимального часу відповіді. Знаходити тренди на панелях продуктивності.

3. Перевірка функціональність API

Ми можемо перевіряти перенаправлення, аутентифікації, виконання CRUD або будь-якого іншого взаємодії API. Відстежувати коди результатів і перевіряти, чи повертає API очікуваний контент.

4. Отримувати сповіщення негайно

Оскільки тести проводяться кожну хвилину, ми будемо знати, коли у API виникають проблеми з оповіщенням і ескалації.

5. Додавання аутентифікації

Ми зможемо отримувати доступ до захищених API, додавши необхідний метод аутентифікації (Basic, NTLM, Digest і OAuth).

6. Відповідність шаблонами RegEx

Використовувати кілька регулярних виразів для налаштування складного зіставлення зі зразком незалежно від того, чи повертає API-відповідь відповіді XML або JSON.

7. Мати повний контроль

Ми будемо мати повний контроль над вмістом HTTP-запиту.

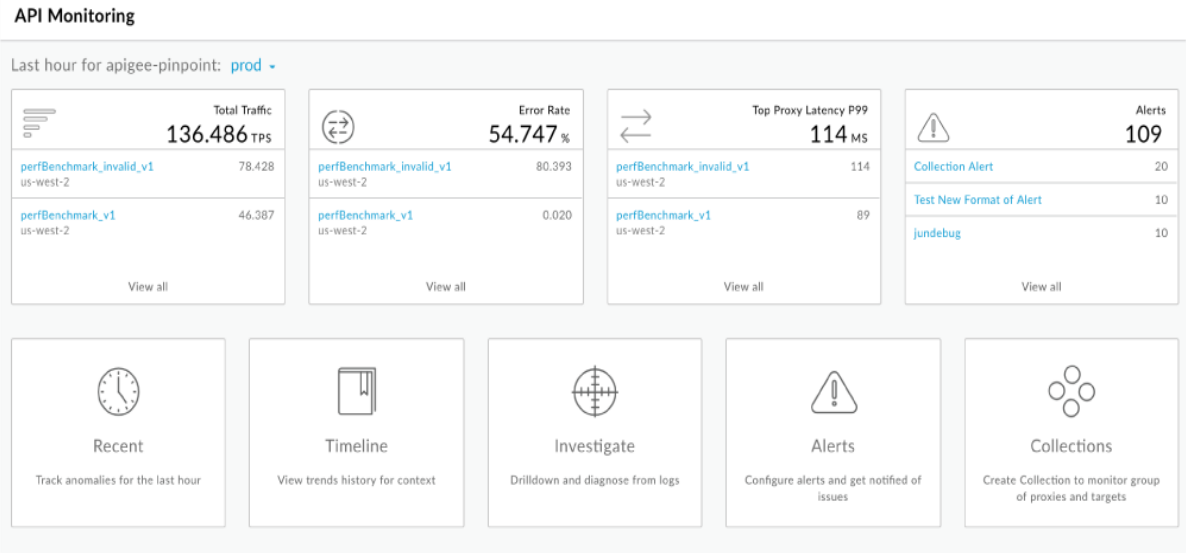


Рис.2.2 Тест API моніторингу

- Доступність сервера.

Поки сервер або пристрій розпізнає протокол TCP / IP, служба моніторингу може перевіряти доступність пристрою і вибраних портів. Служба моніторингу може перевіряти доступність так часто, як раз в хвилину, запобігаючи дорогі простої і втрату продуктивності через Інтернет або за брандмауером.

Існує кілька компонентів, які необхідно ретельно враховувати для забезпечення високої доступності на практиці. Висока доступність залежить не тільки від програмної реалізації, а й від таких факторів, як:

- Навколишнє середовище: якщо всі ваші сервери розташовані в одній географічній зоні, такі умови навколишнього середовища, як землетрус або повінь, можуть пошкодити всю вашу систему. Наявність надлишкових серверів в різних центрах обробки даних і географічних областях підвищить надійність.
- Апаратне забезпечення: сервери високої доступності повинні бути стійкими до перебоїв в живленні і апаратних збоїв, включаючи жорсткі диски і мережеві інтерфейси.
- Програмне забезпечення: весь програмний стек, включаючи операційну систему і сам додаток, повинен бути підготовлений до обробки

несподіваного збою, який може, наприклад, потенційно вимагати перезавантаження системи.

- Дані: втрата даних і неузгодженість можуть бути викликані декількома факторами, і це не обмежується збоями жорсткого диска. Системи високої доступності повинні враховувати безпеку даних в разі збою.
- Мережа: незаплановані перебої в роботі мережі є ще одну можливу точку відмови для високодоступних систем. Важливо, щоб існувала надмірна мережева стратегія для можливих збоїв.

Кожен шар високодоступних системи матиме різні потреби з точки зору програмного забезпечення і конфігурації. Однак на рівні додатків балансувальник навантаження є важливою частиною програмного забезпечення для створення будь-яких налаштувань високої доступності.

HAProxy (проксі високої доступності) є поширеним вибором для балансування навантаження, оскільки він може обробляти балансування навантаження на декількох рівнях і для різних типів серверів, включаючи сервери баз даних.

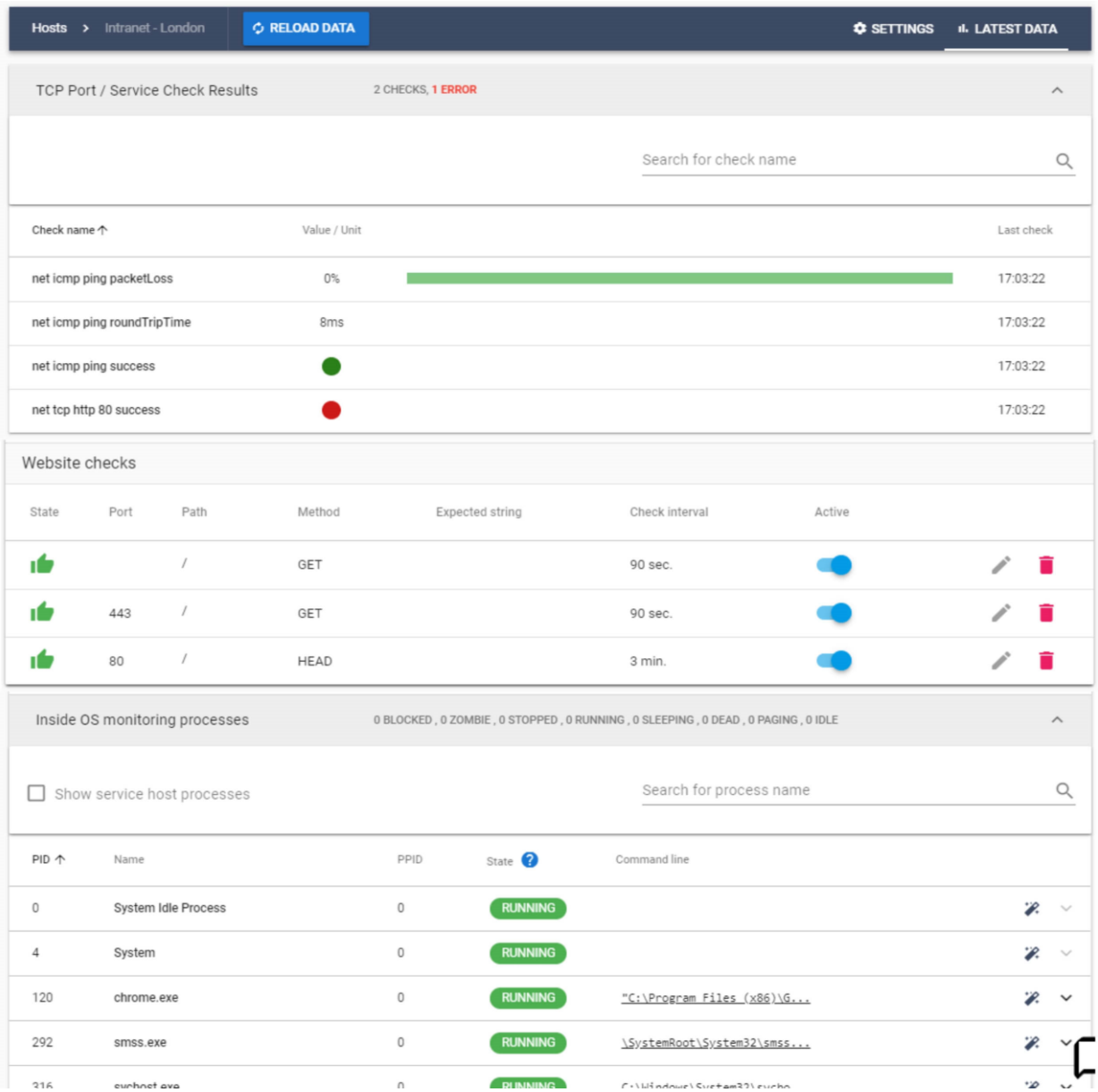


Рис.2.3 Тест доступності сервера

- Розширена доступність

РД- це спеціалізовані автоматизовані монітори перевіряють записи DNS, перевіряють правильність конфігурації сертифікатів SSL, запитують бази даних, входять на поштові сервери та завантажують файли з FTP-серверів.

- DNS

DNS - тип монітора відстежує DNS для стабільності і безвідмовної роботи. Контролюючи свій DNS, ви гарантуєте, що ваша конфігурація DNS залишається такою, як ви планували.

- SSL-сертифікати

Тип монітора SSL-сертифікатів контролює ваші SSL-сертифікати, щоб гарантувати їх постійну доступність і термін їх дії.

- FTP

FTP - тип монітора відстежує FTP - сервер для безперебійної роботи і доступності.

Наступні функції розглядаються як розширені методи перевірки доступності:

- Поєднання основних методів

Ви можете визначити послідовність основних методів для вимоги будь-яким способом.

- Активація ATP на основі правил

Використовуючи зумовлені правила, система автоматично виконує перевірку ATP для продуктів розташування. Тому ми можемо використовувати ATP на основі правил, щоб автоматично реагувати. Ось деякі з доступних опцій:

- Визначення альтернативних місць розташування (визначення місцезнаходження)
- Визначення альтернативних продуктів (заміна продукту)
- Характеристика заміщення
- Визначення альтернативних моделей виробничого процесу (PPM)
- Виклик виробничого і точного планування (PP / DS)
- Здатність до обіцянці

Ми можемо налаштувати автоматичний виклик виробничого планування і докладного планування, коли продукт не повністю доступний.

- Багаторівнева перевірка ATP

Складання на більш низьких рівнях виробництва вже вироблені або заготовлені до надходження замовлення на продаж, але остаточне складання

зініціюється тільки при надходженні замовлення на продаж, саме на моменті остаточного складання потрібна багаторівнева перевірка ATP.

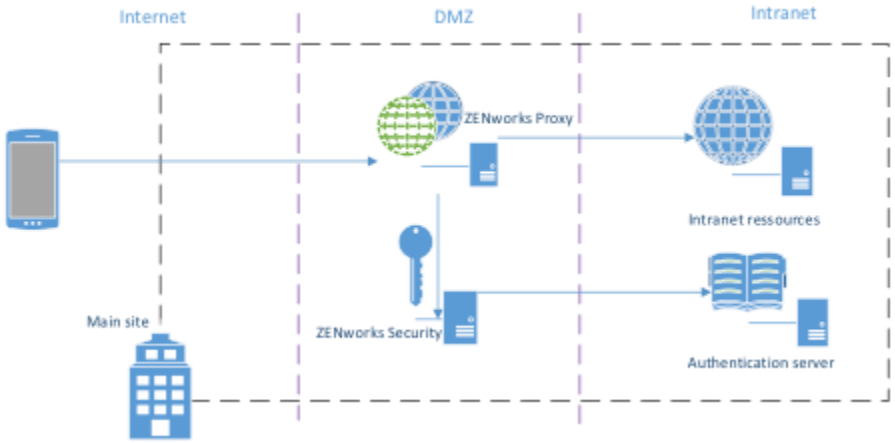


Рис.2.4 Структура ATP

- Моніторинг продуктивності

Моніторинг продуктивності перевіряє швидкість роботи сайту або сервісу. Монітори продуктивності відстежують час для швидкості з'єднання (інтерфейс і бекенда) і час завантаження браузера. Монітори продуктивності можуть використовувати синтетичний моніторинг або технологію RUM. RUM і Full Page Check забезпечують найбільш повний набір даних про продуктивність. Повна перевірка сторінки дає докладні дані про продуктивність для кожного елемента на сторінці. Монітори продуктивності видають попередження про помилки сторінок, відсутньому контенті і низьку продуктивність.

- Повний моніторинг завантаження сторінки

З зростаючим попитом на швидкість ми повинні знати, який елемент, коли і де стає showstopper. За допомогою моніторингу повного завантаження сторінки ми отримаємо інформацію про час завантаження кожного елемента за кожну годину.

Грунтуючись на статистичному аналізі часу повного завантаження сторінки, ми можемо передбачити подальшу поведінку обладнання і, отже,

зможемо планувати оновлення обладнання без шкоди для продуктивності сторінки.

- Моніторинг синтетичних транзакцій

Іншим важливим фактором є функціональна перевірка веб-сторінок. Ми повинні регулярно тестувати і виконувати ті ж дії, що і користувачі. Це займе багато часу. Замість цього ми можемо використовувати моніторинг синтетичних транзакцій. Моніторинг синтетичних транзакцій використовує зумовлений сценарій для виконання тих же дій, що і користувачі. Ми можете легко записати сценарій за допомогою реєстратора транзакцій, а потім налаштувати монітор, який буде використовувати записаний сценарій.

Скрипт буде імітувати потік веб-транзакцій і перевіряти функціональність кожного кроку.

- Моніторинг реальних користувачів

Real User Monitoring (RUM) збирає дані від реальних користувачів, щоб дати вам уявлення про продуктивність вашого сайту.

RUM збирає призначені для користувача дані про перегляди сторінок вашого веб-сайту і часу завантаження, а також про те, де цей час витрачається, починаючи з моменту, коли користувач вводить адресу сторінки або натискає на посилання, до тих пір, поки веб-сторінка не буде повністю завантажена і відображена для користувача.

- Функціональний моніторинг

Такий процес, як функціональний моніторинг, дозволяє спостерігати функціональні можливості, які може запропонувати додаток або розподілена система. Її головна мета - оцінити доступність і функціональність системи і можливість її виконання бажаними функціями.

Функціональний моніторинг виконується автоматично з використанням машин і різних скриптів. Оскільки цей тип моніторингу заснований на роботі роботів, він підходить для управлінських звітів, які перевіряють якість обслуговування, яке відчувають користувачі системи. Функціональний

моніторинг заснований на специфікаціях програмного компонента. За допомогою цього процесу можна контролювати відповідність кожної функції програми певним вимогам.

2.3. Серверна частина системи

Виконуючи збір метрик веб-сервіса, необхідна можливість зберігати великі обсяги даних, постійно записувати нові дані та швидко їх діставати за запитом.

Зберігати дані на самому сервері це невдале рішення, таку систему дуже важко масштабувати, тому я обрав для даної цілі базу даних MongoDB [14].

MongoDB зберігає дані у BSON форматі, він подібний до JSON. Структура даних є гнучкою, тобто нові поля можна просто писати у об’єкт та ніяких міграцій не потрібно. MongoDB – це відкрита та безкоштовна кросплатформенна розподілена база даних, зберігає дані у форматі ключ-значення, підтримує горизонтальне масштабування та багато видів індексів.

Основні переваги MongoDB:

1. Продуктивність.
2. Гнучкість, легко змінюється структура даних документа.
3. Швидкі та зручні індекси.
4. Масштабованість.

Для серверної частини я обрав Node.js. Це відкрита платформа для виконання високопродуктивних серверних застосунків. Платформа Node.js [12] дозволяє писати мережеві застосунки на мові JavaScript, з великою спільнотою розробників та готових модулів, та бібліотек. Для виконання JavaScript коду використовується двигун V8 розроблений компанією Google. Основні властивості Node.js: однопотокова та асинхронна модель виконання логіки, неблокуючий ввід/вивід, легко масштабуємий застосунок.

Основні переваги вибору:

1. Асинхронний принцип роботи.

2. Швидкість.
3. Принцип однопотокового написання бізнес логіки.
4. Легке масштабування застосунку.

Інфраструктура Node.js включає в себе менеджер пакетів npm, для інсталяції, оновлення, оновлення, аудиту та перевірки безпечності використаних модулів.

Для серверів побудованих за допомогою Node.js [10] досить часто використовується фреймворк Express.

Філософія Express полягає у наданні невеликих, надійних інструментів для HTTP-серверів, що робить його чудовим рішенням для додатків для однієї сторінки, веб-сайтів, гібридів або публічних HTTP-API. Інсталювати бібліотеку можна за допомогою команди npm: npm install express –save

Корисні функції даної бібліотеки:

- Дозволяє гнучко комбінувати лоігку сервісу за допомогою middleware
- Надійна, швидка та гнучка маршрутизація
- Висока продуктивність
- HTTP помічники (перенаправлення, кешування тощо)
- Підтримує понад 14 шаблонів для рендерингу

2.4. Клієнтська частина системи

Я обрав JavaScript мовою програмування для реалізації клієнтської частини програми. Будь-який сайт складається з розмітки HTML, стилів CSS та клієнтського коду застосунку на JavaScript.

JavaScript це мова, яка дозволяє застосовувати складні речі на web сторінці - кожен раз, коли на web сторінці відбувається щось більше, ніж просто її статичне відображення - відображення періодично оновлюваного контенту, або інтерактивних карт, або анімація 2D / 3D графіки, або прокрутка відео в програвачі, і т.д.

JavaScript – це інтерпретована мова, використовується як мова для веб сценаріїв сторінок. Можливість швидко створювати прототипи, швидкодія

обрахунків та велика кількість різноманітних модулів перенесли мову з клієнтської частина на серверну у вигляді платформи Node.js [9].

Основні переваги вибору мови JavaScrip:

- універсальність, підтримується всіма браузерами;
- синтаксис є С-подібним, тобто зрозумілим та простим;
- через свою популярність, має дуже багато готових функцій та бібліотек;
- асинхронний принцип роботи;
- управління контентом сторінки та можливість створення динамічних даних;

Насправді використання JavaScript може бути значно ширшим, ніж тільки для web-сторінок. За допомогою цієї мови програмування можна створювати скрипти, що автоматизують будь-які рутинні операції, які можна не виконувати вручну, а перекласти на плечі програми.

HTML та CSS – це основа будь-якого сайту. HTML це мова розмітки, яку ми використовуємо для візуального і смислового структурування нашого web контенту, тобто, визначаємо параграфи, заголовки, таблиці даних, або вставляємо зображення і відео на сторінку. CSS це мова стилів за допомогою якої ми надаємо стиль відображення нашого HTML контенту, наприклад надаємо колір фону (background) і шрифту, надаємо контенту багато колонного вигляду.

Висновки до розділу 2

В даному розділі я розглянув принцип та методи проектування системи збору та аналізу метрик сервера. Стрімкий розвиток веб технологій робить систему збору на аналізу метрик сервера потрібною. Даний аналіз доводить, що створення даного застосунку є доцільно. Дослідження не показало аргументів щодо неможливості створення даного веб сервісу.

Для побудови застосунку я обрав клієнт-сервісну архітектуру з тонким клієнтом.

Технології для реалізації:

1. MongoDB – нереляційна база даних.
2. Node.js – серверна платформа для розробки.
3. JavaScript – клієнтська скриптова мова розробки.
4. HTML + CSS – основні засоби для створення клієнтського інтерфейсу.
5. Express [11] – бібліотека для зручного написання серверної логіки.

РОЗДІЛ 3

РЕАЛІЗАЦІЯ СИСТЕМИ ЗБОРУ ТА АНАЛІЗУ МЕТРИК СЕРВЕРА

3.1 Аналіз метрик даних сервера

Багато різних факторів впливають на ефективність, популярність, гнучкість та надійність веб сервера:

- Сервер дуже навантажений (зараз багато клієнтів або щось пішло не так).
- Не можливо визначити ір-адресу сервера – проблема з DNS.
- Великий час відповіді сервера – виконання складних задач чи велика кількість клієнтів у даний момент.
- Проблема з базою даних – втрачено зв’язок чи якісь проблеми з даною частиною системи.
- Постійний перезапуск сервера через невірне конфігурування чи через не достаток ресурсів сервера.
- Виконання запланованих раніше періодичних завдань може призвести до зменшення продуктивності сервера в цей момент чи навіть не доступним.
- Неможливість запису даних на сервер через закінчення вільного місця на носію.
- Завелика кількість операцій запису та зчитування з фізичного носія.
- Троттлінг процесора сервера через велику температуру нагріву фізичних компонентів сервера.

При перевірці веб сервера необхідні такі дані:

id – унікальний номер перевірки

service_name – назва сервісу

server_number – унікальний номер сервера

request_time – час надсилання запиту на сервер

uptime – час роботи сервера

cpu_speed – швидкість роботи процесора (у Герцах)

cpu_load – навантаження на процесор

cpu_temperature – температура процесора

mem_total – об’єм максимальної можливої оперативної пам’яті сервера

mem_free – об’єм вільної оперативної пам’яті сервера

mem_used - об’єм використаної оперативної пам’яті сервера

mem_active - об’єм активної оперативної пам’яті сервера

mem_swap_total - розмір своп-пам’яті

mem_swap_used - об’єм використаної своп-пам’яті сервера

mem_swap_free - об’єм вільної своп-пам’яті сервера

platform – платформа операційної системи

last_login_user_name – ім’я юзера, який останній підключався до сервера

last_login_user_date – час, коли останній раз підключались до сервера

last_login_ip_address – адреса, з якої останній раз підключались до сервера

average_load – середня навантаження на сервер за останні 30 хвилин

running_processes_number – кількість запущених процесів на сервері

disk_total – максимальний об’єм пам’яті сервера

disk_free - максимальний об’єм вільної пам’яті сервера

disk_used - максимальний об’єм вільної пам’яті сервера

disk_system_type – тип файлової системи

open_files_number – кількість відкритих файлових дескрипторів

io_load – відсоток навантаження на диск операціями запису та зчитування

network_stats – навантаження мережевої карти

creation_time – час коли були створені дані

3.2 Серверна частина

У таблиці 3.1 наведені HTTP методи, для передачі та отримання даних сервера.

Таблиця 3.1

Метод запиту HTTP	URL адреса	Параметри	Призначення
1	2	3	4
PUT	/users/:id	id: ідентифікатор користувача first_name: (optional) ім’я користувача sur_name: (optional) прзівище користувача password: (optional) пароль користувача	Оновити дані про користувача
GET	/users/:id	id: ідентифікатор користувача	Отримати дані про користувача
DELETE	/users/:id	id: ідентифікатор користувача	Видалити дані про користувача
POST	/users/login	email: електронна пошта користувача password: пароль користувача	Аутентифікувати юзера, повернути токен, для подальшої роботи клієнта з методами сервера

Таблиця 3.1(продовження)

1	2	3	4
POST	/users/signup	first_name: ім’я користувача sur_name: прзівище користувача email: електронна пошта користувача password: пароль користувача	Створити користувача та повернути токен для подальшої аутентифікації користувача
POST	/services	?user_token: токен для ідентифікації користувача service_name: назва сервісу server_address: адрес сервера для перевірки period: час через який опитувати сервіс	Створити сервіс користувача
GET	/services	?user_token: токен для ідентифікації користувача	Отримати всі сервіси користувача
GET	/services/:service_id	service_id: ідентифікатор сервіса ?user_token: токен для ідентифікації користувача	Отримати дані про сервіс користувача

Таблиця 3.1(продовження)

1	2	3	4
PUT	/services/:service_id	service_id: ідентифікатор сервіса ?user_token: токен для ідентифікації користувача service_name: (optional) назва сервісу server_address: (optional) адрес сервера для перевірки period: (optional) час через який опитувати сервіс	Оновити дані про сервіс користувача
DELETE	/services/:service_id	service_id: ідентифікатор сервіса ?user_token: токен для ідентифікації користувача	Видалити дані про сервіс користувача
GET	/checkups	?user_token: токен для ідентифікації користувача ?service_id: ідентифікатор сервісу користувача	Отримати список всіх перевірок користувача для певного сервісу
GET	/checkups/coming	?user_token: токен для ідентифікації користувача ?service_id: ідентифікатор сервісу користувача	Отримати список перевірок, що скоро будуть опитані

Таблиця 3.1 (закінчення)

1	2	3	4
GET	/checkups/done	?user_token: токен для ідентифікації користувача ?service_id: ідентифікатор сервісу користувача ?timeFrom: фільтр по часу коли сталася перевірка ?timeTo: фільтр по часу до поки сталася перевірка ?size: кількість перевірок які хочу отримати ?offset: кількість перевірок, які потрібно пропустити	Отримати список перевірок, які вже завершені для певного сервісу
GET	/checkups/:id	id: ідентифікатор перевірки ?user_token: токен для ідентифікації користувача ?service_id: ідентифікатор сервісу користувача	Отримати перевірку
PUT	/checkups/:id/	id: ідентифікатор перевірки ?user_token: токен для ідентифікації користувача ?service_id: ідентифікатор сервісу користувача Дані описані у пункті 3.1	Записати дані перевірки

3.3 **Клієнтська частина**

Під час розробки клієнтської частини потрібно зконфігурувати проект. Оскільки я використав платформу Node.js, вона містить пакетний менеджер npm. Завдяки ньому я можу інсталювати пакети з хмарного сервера npm на потрібний мені локальний сервер. Для ініціалізації потрібно запустити команду: npm init. В результаті я отримаю файл package.json який включатиме в себе назву проекту, його автора, версію, залежності, залежності для розробки і так далі.

Встановити всі залежності можна можна командою: npm install. Після цього у проекті буде створена папка node_modules, яка міститиме в собі всі залежності та їх можна буде використовувати у проекті.

Встановити певну залежність можна використовуючи команду: npm install [package], де package – це назва бібліотеки, яку ми хочему інсталювати.

У таблиці 3.2 я навів весь список залежностей проекту для системи моніторінгу та аналізу метрик сервера.

Таблиця 3.2

Назва	Версія	Призначення
1	2	3
mongoose	5.9.16	Асинхронний інструмент для відображення та поєднання даних та моделей об’єктів MongoDB.
express	4.17.11	Бібліотека для NodeJS, яка забезпечує швидкість розробки та гнучкість налаштування проекту веб сервера
body-parser	1.18.2	Плагін, для аналізу та розбиття тіла HTTP запросу

Таблиця 3.2(закінчення)

1	2	3
express-validator	5.0.11	Плагін, для валідації вірності надісланих даних.
mongoose-paginate-v2	1.3.9	Плагін, для підтримки сторінок під час запиту даних з MongoDB.
mongoose-auto-increment	5.0.1	Плагін, для створення нумерованого поля, яке буде самостійно збільшуватись та завжди буде унікальним,
pug	3.0.0	Мова шаблонів для генерації HTML-розмітки з вкрапленням JavaScript.
config	3.3.1	Допомагає гнучко налаштувати систему під час запуску, вказати параметри запуску, внести параметри за замовчунням, параметри для різних середовищ.
morgan	1.10.0	Плагін для запису даних про отримані сервером запити.
ws	7.3.0	Імплементація сервера та клієнта протоколу WebSocket для швидкого обміну даними над HTTP.
helmet	3.22.0	Плагін для запобігання вразливості сервера, контролем небезпечних HTTP хедерів.
moment	2.26.0	Бібліотека для зручного та гнучного управління, аналізу, перевірки, зміни, форматування дат та часу.
nodemailer	6.4.8	Бібліотека для надсилання електронних емейлів потрібим користувачам з вказаним текстом.

Основні функції мого веб-сервісу:

- Забезпечувати роботу веб сервісів
- Зібрати метрики серверів
- Провести аналіз зібраних метрик
- Зберегти дані для подальшої звітності веб-сервісу
- Повідомити користувача у разі помилки чи можливої небезпеки
- Зручний та зрозумілий інтефейс
- Зручне та гнучне налаштування принципу збору метрик
- Надати повну інформацію про перевірки, особливо про невдалі
- Гнучкий та швидкий API сервера для подальшого розвитку проекту
- Доступ до API сервера для подальшої інтеграції інших сервісів
- Можливість створення власних плагінів
- Відсутня інсталяція та дуже легке адміністрування

На рис. 3.1 показано структуру бази даних системи:

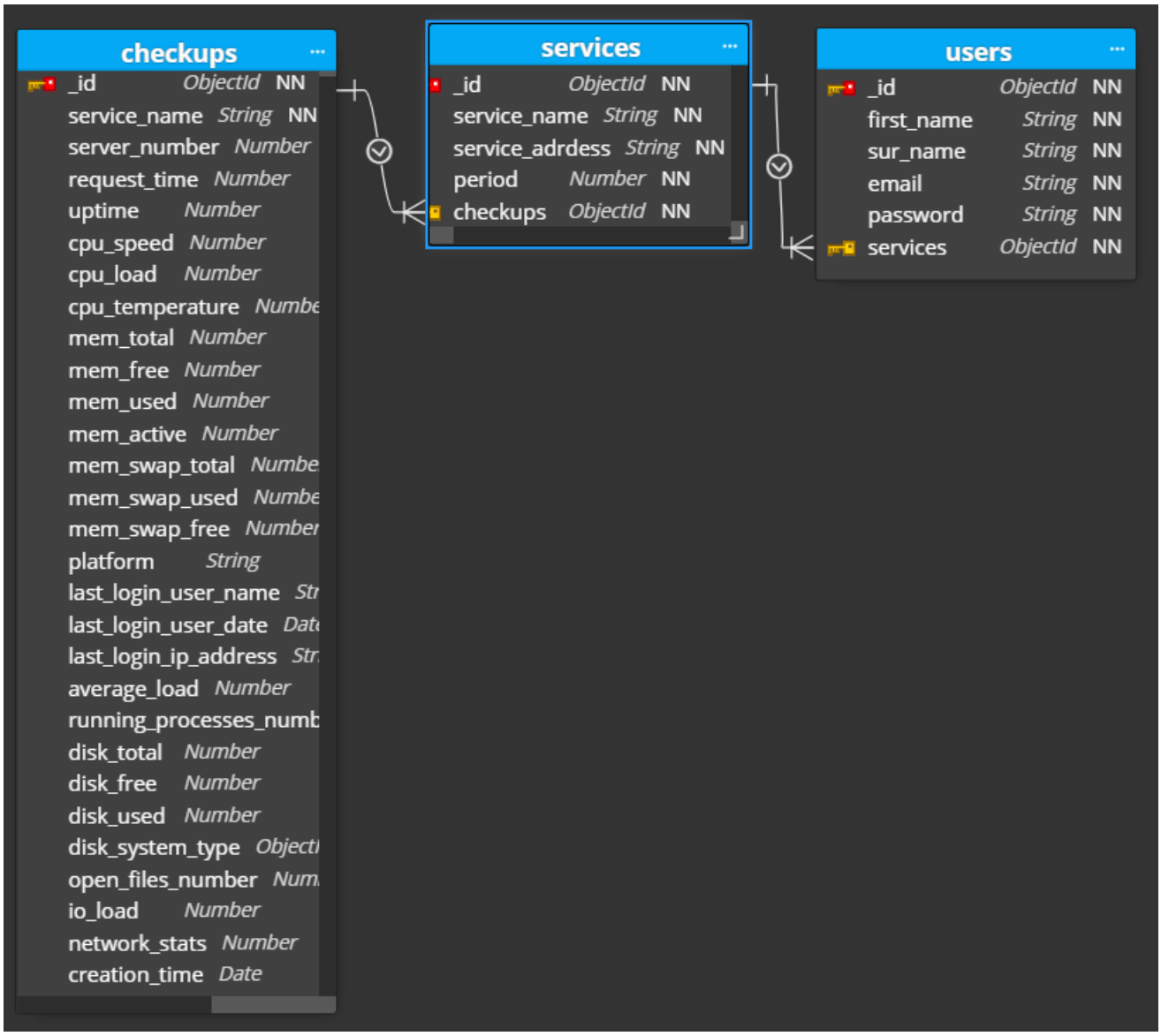


Рис. 3.1 База даних системи

3.4 Інструкція користувача

Основні сторінки клієнтського інтерфейсу та їх функціонал: Звичайно першою сторінкою є сторінка авторизації (рис. 3.2)

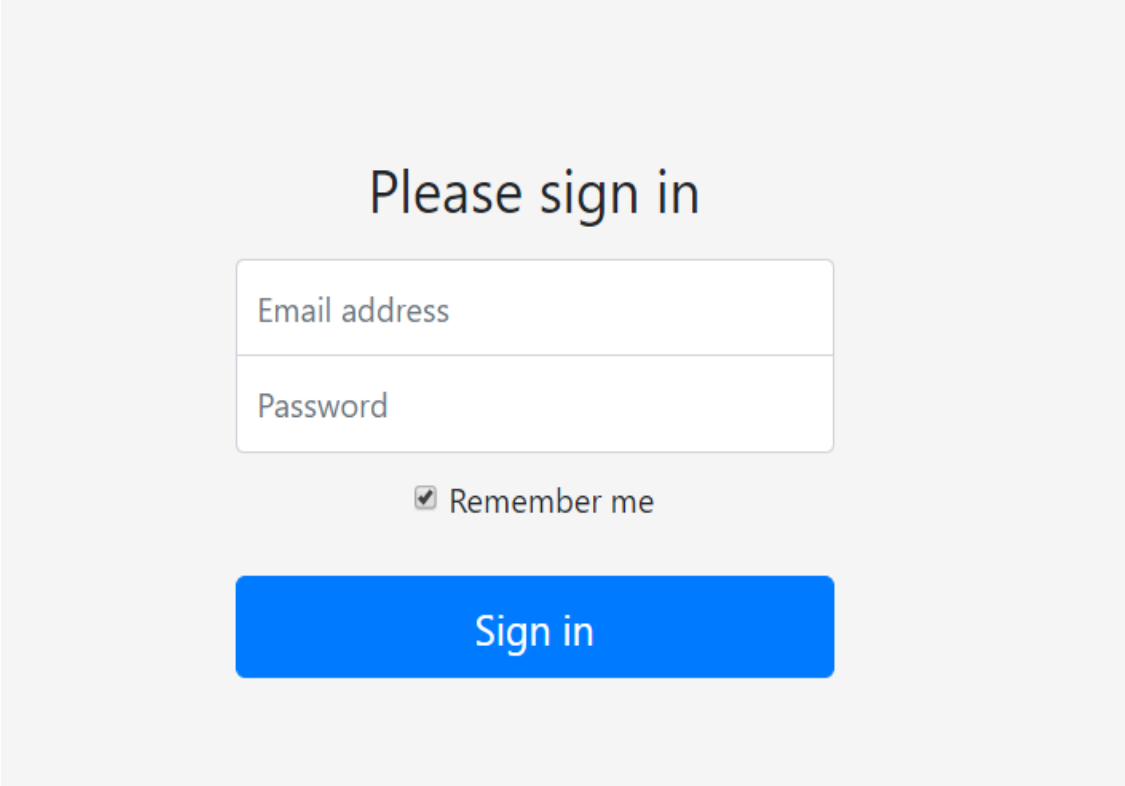


Рис. 3.2 Сторінка авторизації

Якщо користувач не зареєстрований, то йому потрібно це зробити на відповідній сторінці (рис. 3.3)

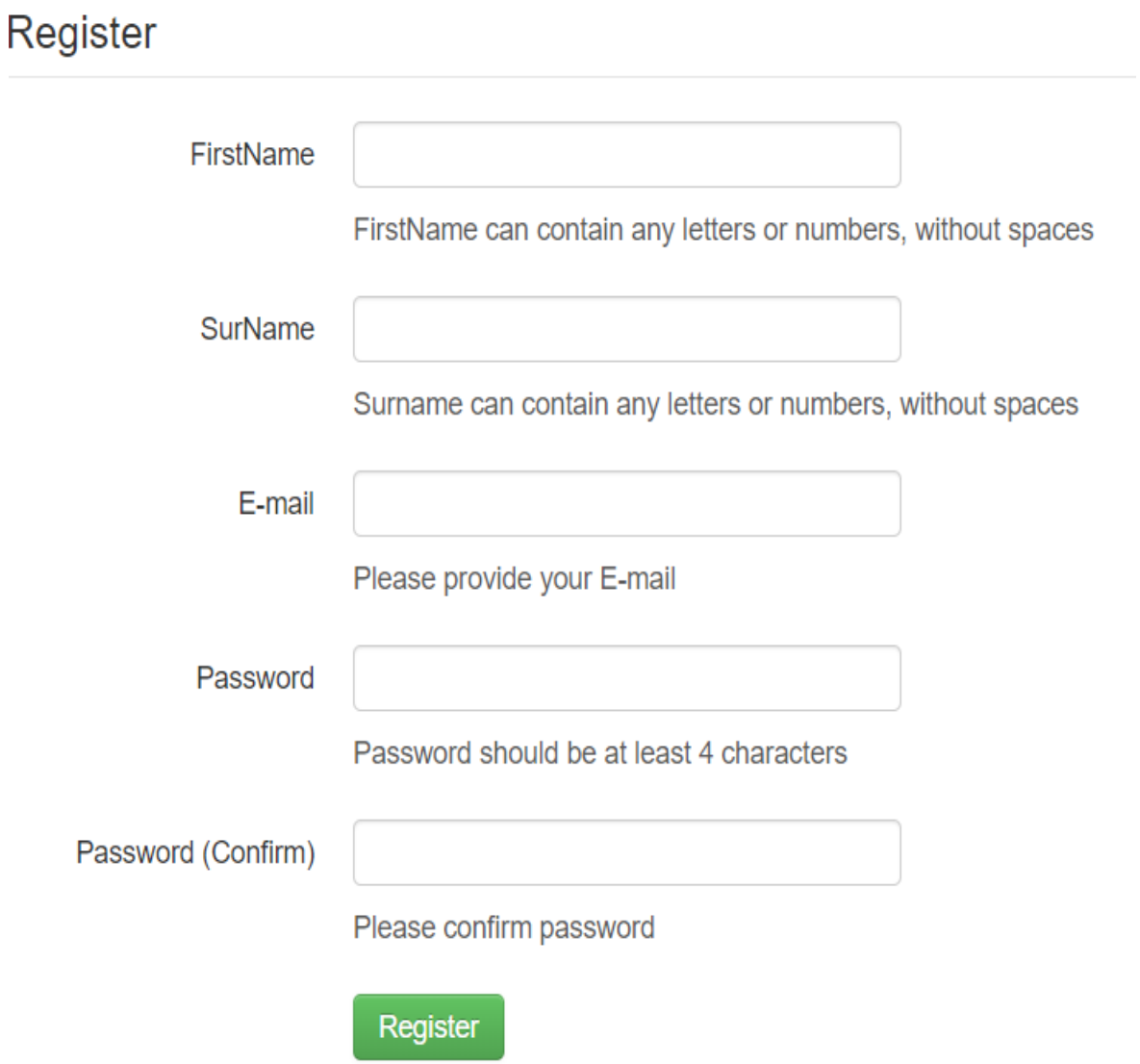


Рис. 3.3 Сторінка реєстрації

Після авторизації першою сторінкою є сторінка з дашбордом (рис. 3.4)

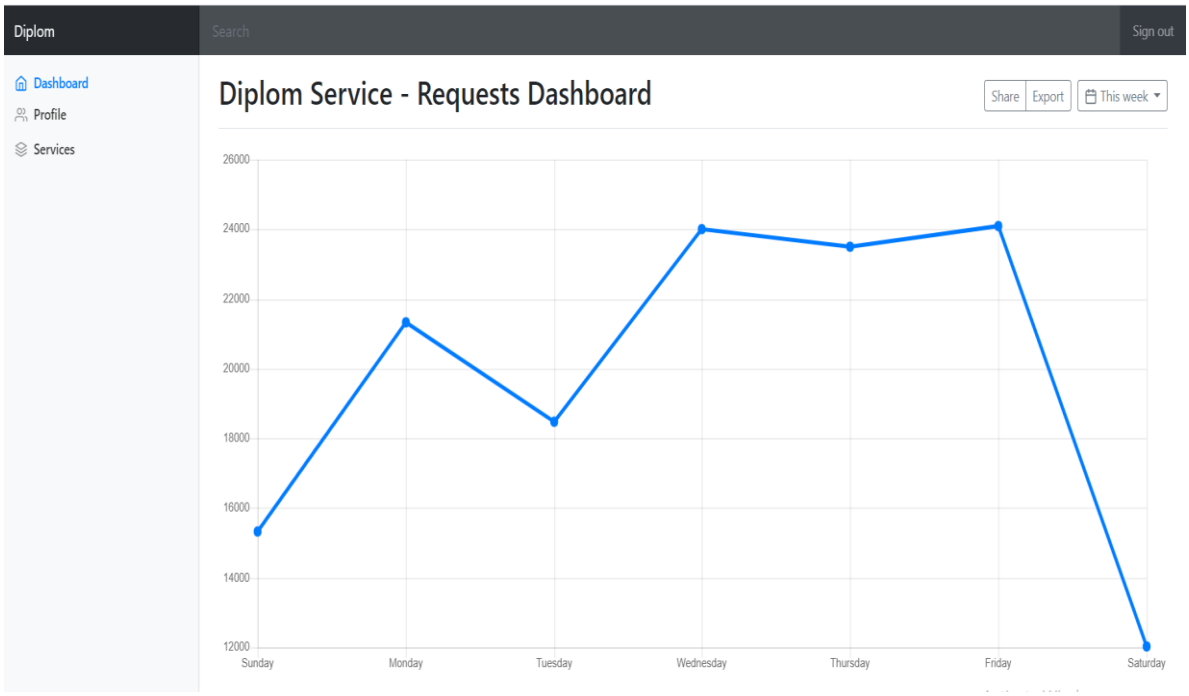


Рис. 3.4 Сторінка дашборду

Для створення перевірки сервісу потрібно перейти по “Services”, потім вибрати “Create New”. Потім на сторінці (рис. 3.5) створити перевірку сервісу.

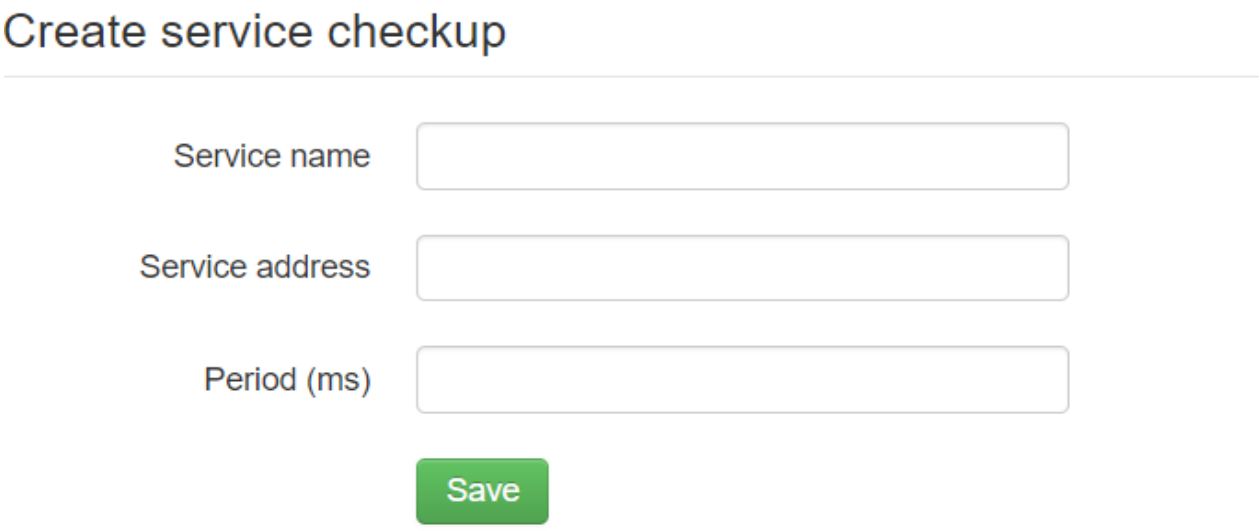


Рис. 3.5 Створення перевірки сервісу

Для перегляду всіх перевірок, їхнього статусу, потрібно натиснути кнопку “Checkups” (рис. 3.6)

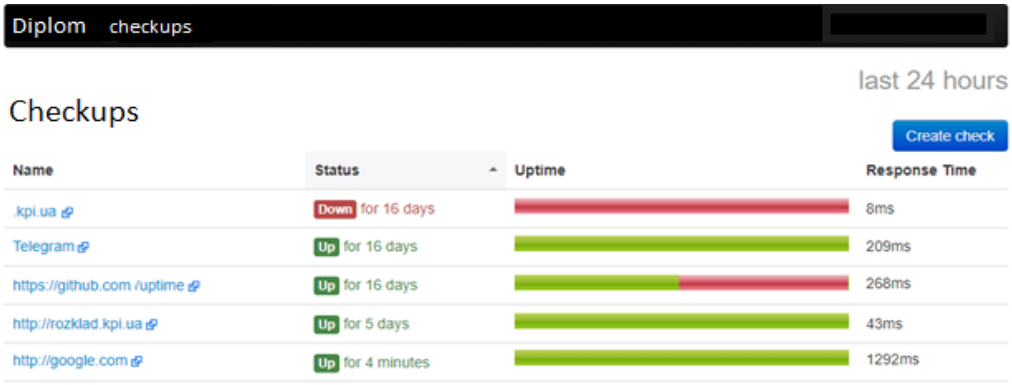


Рис. 3.6 Перегляд стану перевірок

Перегляд активності (рис. 3.7), часу відповіді (рис. 3.8), завантаження процесора (рис 3.9), завантаження диску (рис 3.10), температури процесора (рис 3.11), вільної оперативної пам’яті (рис 3.12), середнє навантаження сервера (рис 3.13), навантаження мережевої карти (рис 3.14).

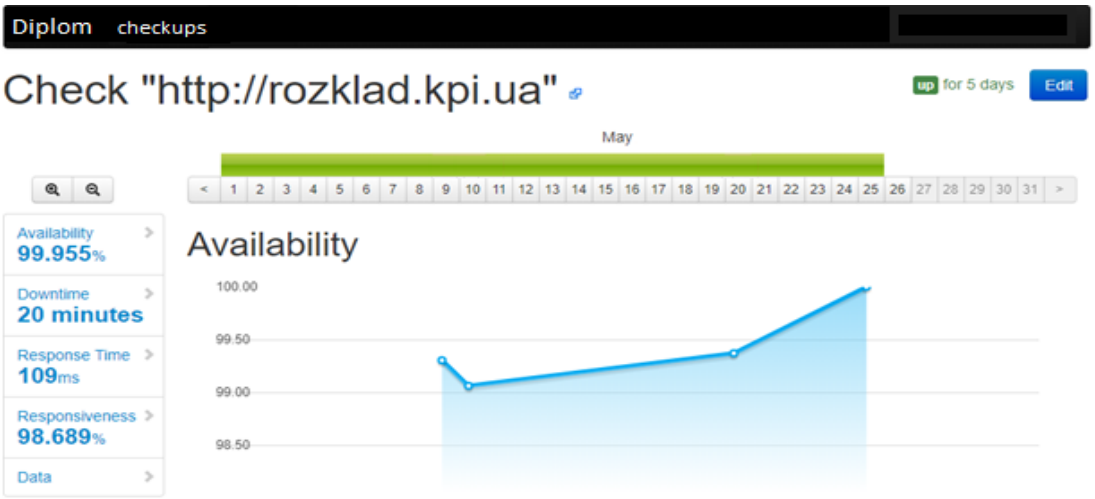


Рис. 3.7 Активність сервісу

Рис. 3.8 Час відповіді

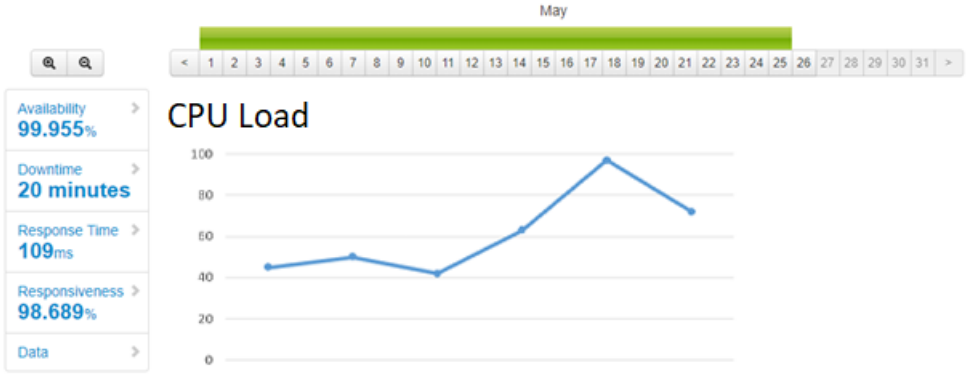


Рис. 3.9 Завантаження процесора

Рис. 3.10 Завантаження диску

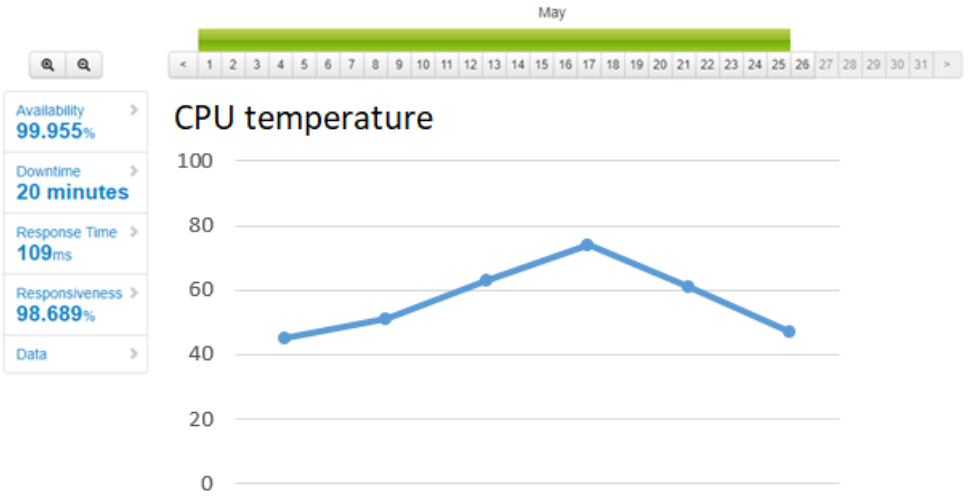


Рис. 3.11 Температура процесора

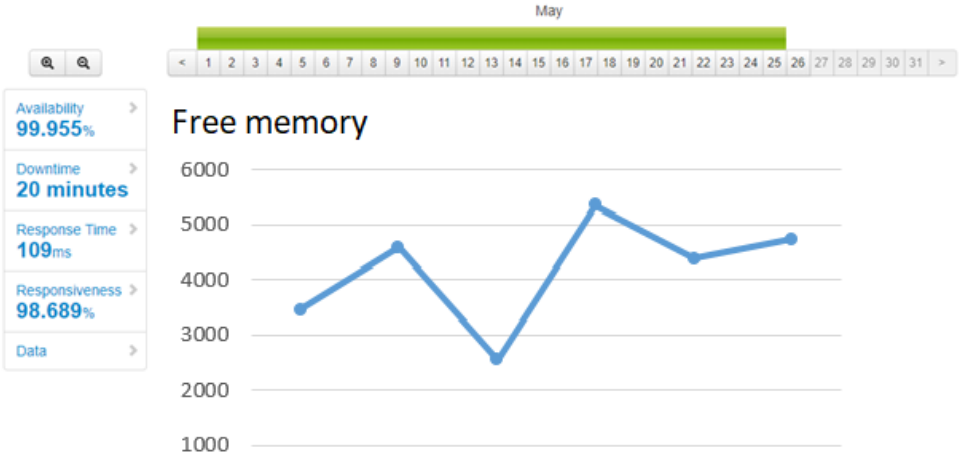


Рис. 3.12 Вільна оперативна пам’ять

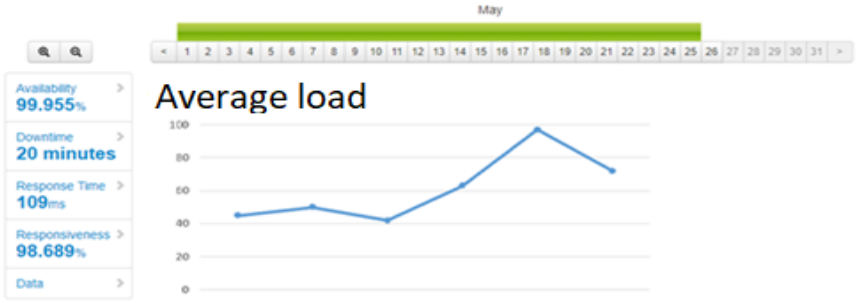


Рис. 3.13 Середнє навантаження сервера

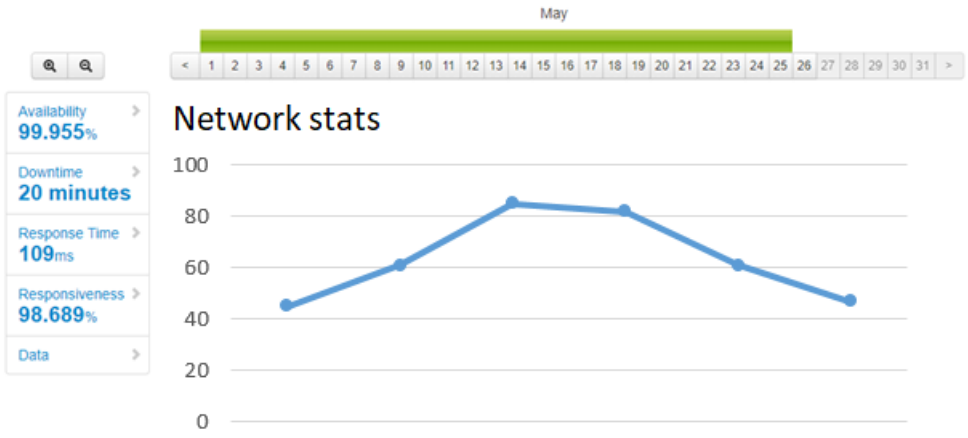


Рис. 3.14 Навантаження мережевої карти

Висновки до розділу 3

В даному розділі описані принципи створення архітектури, побудови та реалізації системи збору та аналізу метрик сервера. Для реалізації веб сервера використано Node.js, Express, MongoDB та Mongoose. Для реалізації клієнтської частини використано HTML, CSS, JavaScript, Bootstrap.

Система була розділена на окремі модулі для гнучкого розвитку сервісу надалі та легкої підтримки системи. Така реалізаціє є більш гнучкою та надійною, через можливість вільного тестування системи. Дана реалізація точно слідує принципу DRY.

Клієнтський інтерфейс гарний, зручний та інтуїтивно зрозумілий, що дозволяє користувачу без гідного досвіду у програмуванні спокійно використовувати сервіс. Такий підхід дозволить користувачу максимально швидко налаштувати систему під себе.

ЗАГАЛЬНІ ВИСНОВКИ

Метою дипломної роботи була розробка системи збору та аналізу метрик сервера. Система була розроблена мовою програмування JavaScript за допомогою платформи Node.JS та допоміжних бібліотек.

Популярність інтернету сприяла появі великої кількості веб сервісів та веб додатків, за якими потрібно постійно слідкувати та підтримувати, щоб клієнти були задоволені та підтримували сервіс далі. Порівняльний аналіз та опис було наведено у першому розділі дипломної роботи. Потім я виявив декілька проблем у всіх наявних системах, тобто: складність налаштування, обмеженість безкоштовних версій, складність у використанні, повністю ручне налаштування.

Наступний розділ містить детальний опис програмних підходів та засобів для побудови системи, щоб вона відповідала поставленим вимогам та задачам, була гнучкою та продуктивною.

Останній розділ пояснює особливості проектування програми, специфіку та інструкцію використання. Наведена структура бази даних системи.

Отже, наведені у даному дипломному проекті проблеми систем збору та аналізу метрик сервера були вирішені, а також з готовим сервісом забезпечує безпеку потрібних веб-сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alastair Walker Systems, Software and Services Process Improvement / R.V. O'Connor, R.C. Messnarz. – Edinburg, UK: Springer, 2019. – 511 p
2. Icinga [Електронний ресурс] – Режим доступу до ресурсу: https://icinga.com/
3. Nagios [Електронний ресурс] – Режим доступу до ресурсу: https://www.nagios.org/
4. Zabbix [Електронний ресурс] – Режим доступу до ресурсу: https://www.zabbix.com/ru
5. Prometheus [Електронний ресурс] – Режим доступу до ресурсу: https://prometheus.io/
6. Netdata [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/netdata/netdata
7. HTTP [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/215117/
8. CSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Web/Guide/CSS/Getting_started/What_is_CSS
9. Architecture on Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://blog.logrocket.com/the-perfect-architecture-flow-for-your-next-node-js-project/
10. Ethan Brown Web Development with Node and Express – O'Reilly Media, Inc. 2016 – 192 p
11. What is Express [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/ru/
12. Building app with Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://medium.com/devschacht/node-hero-chapter-1-239f7afeb1d1

13. Gathering data principle - [Електронний ресурс] – Режим доступу до ресурсу: https://www.digitalocean.com/community/tutorials/gathering-metrics-from-your-infrastructure-and-applications
14. What is MongoDB? [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/what-is-mongodb

ДОДАТОК А
Система збору та аналізу метрик сервера

СХЕМА СИСТЕМИ

Аркушів 1

Київ 2020 р.

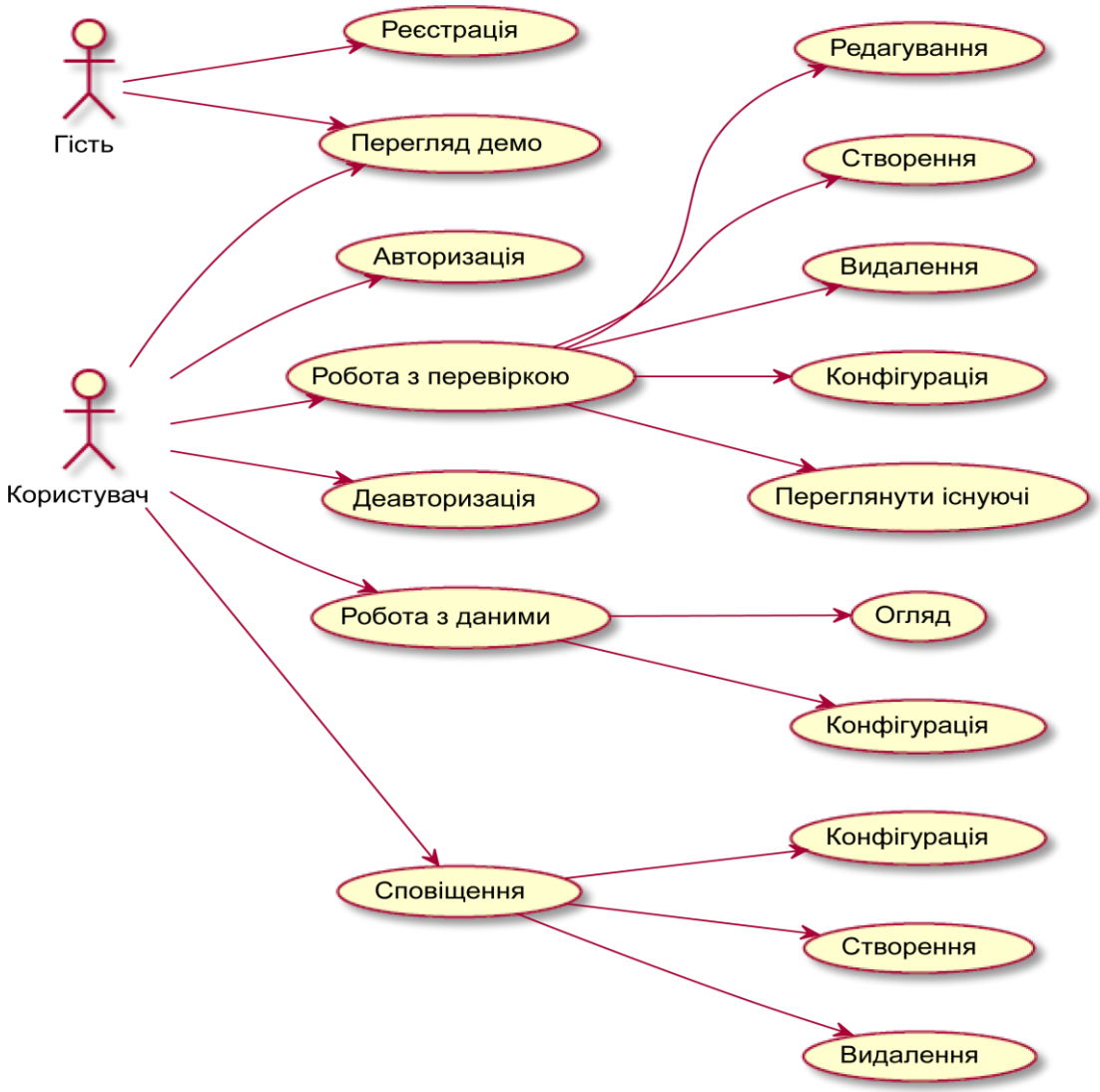
Зм.	Арк.	Прізвище	Підпи	Дата	ІАЛЦ 467200.004 Д1		
Розроб.		Павленко В.М.			*Система збору та аналізу метрик сервера. Додаток*	*Літ.*	*Арк.* / *Аркушів*
Перевірив.		Корочкін О.В.					1
Н. кон.		Сімоненко В.П.			*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, кафедра ОТ гр. ІП-62*		
Затв.		Стіренко С.Г.					

ДОДАТОК Б
Система збору та аналізу метрик сервера

ДІАГРАМА ВАРІАНТІВ ВИКОРИСТАННЯ СИСТЕМИ

Аркушів 1

Київ 2020 р.



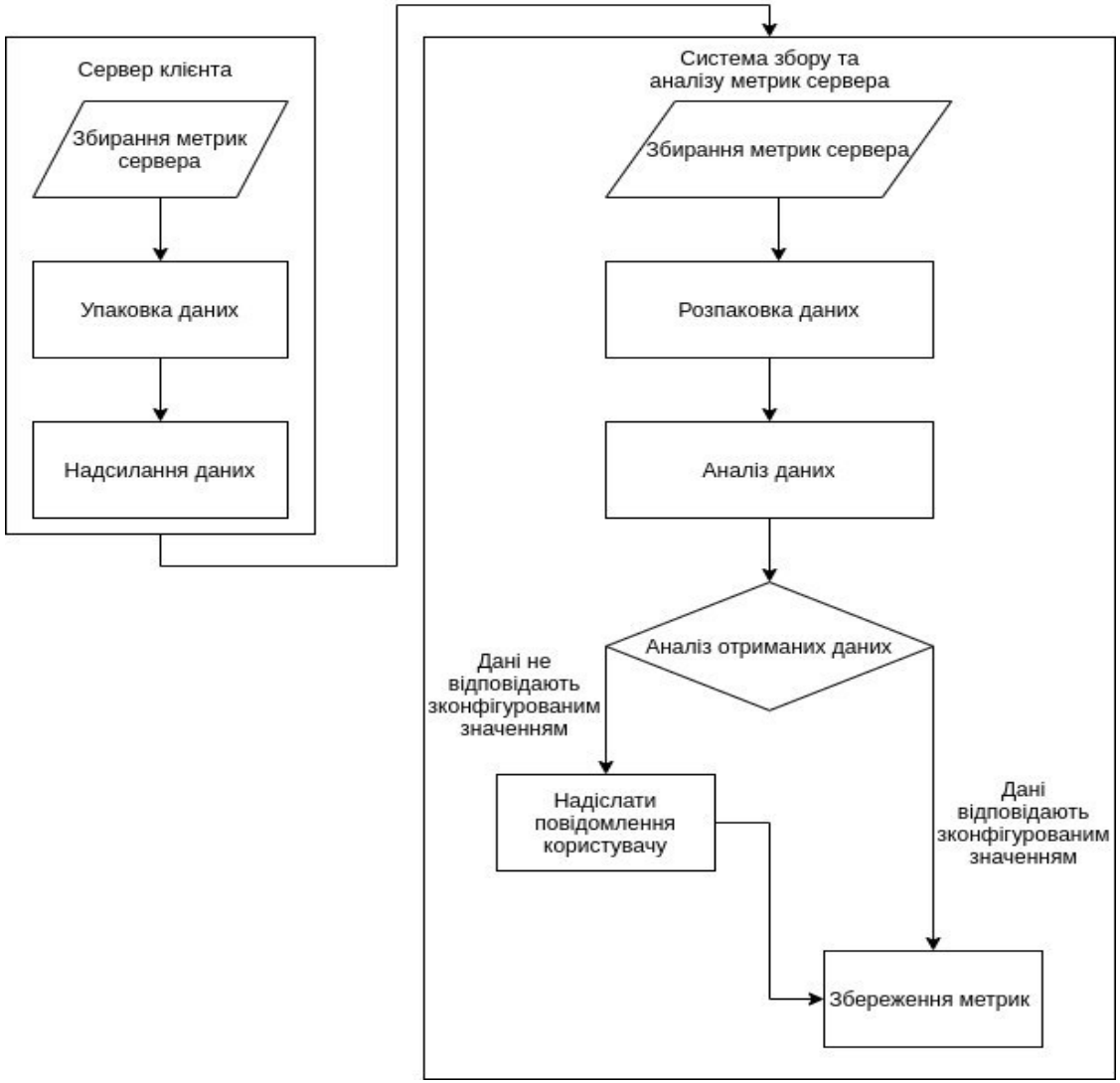
					ІАЛЦ 467200.005 Д2	
Зм.	Арк.	Прізвище	Підп	Дата		
Розроб.		Павленко В.М.			*Система збору та аналізу метрик сервера. Додаток*	*Літ.* | *Арк.* | *Аркушів* 1
Перевірив.		Корочкін О.В.				
Н. кон.		Сімоненко В.П.				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, кафедра ОТ гр. ІП-62*
Затв.		Стіренко С.Г.				

ДОДАТОК В
Система збору та аналізу метрик сервера

АЛГОРИТМ РОБОТИ СИСТЕМИ

Аркушів 1

Київ 2020 р.



					ІАЛЦ 467200.006 Д3	
Зм.	Арк.	Прізвище	Підп	Дата		
Розроб.		Павленко В.М.			*Система збору та аналізу метрик сервера. Додаток*	*Літ.* | *Арк.* | *Аркушів* 1
Перевірив.		Корочкін О.В.				
Н. кон.		Сімоненко В.П.				*НТУУ «КПІ ім. Ігоря Сікорського», ФІОТ, кафедра ОТ гр. ІП-62*
Затв.		Стіренко С.Г.				

ДОДАТОК Г
Система збору та аналізу метрик сервера

ЛІСТИНГ ПРОГРАМИ

Аркушів 39

Київ 2020 р.

/api/registerUser.js

```
async ({ login, password, fullName }) => {
  const hash = await application.security.hashPassword(password);
  await application.auth.registerUser(login, hash, fullName);
  return { result: 'success' };
};
```

/api/resmon.js

```
async () => {
  const loadavg = api.os.loadavg();
  const stats = application.resmon.getStatistics();
  const { heapTotal, heapUsed, external, contexts } = stats;
  const total = application.utils.bytesToSize(heapTotal);
  const used = application.utils.bytesToSize(heapUsed);
  const ext = application.utils.bytesToSize(external);
  return { total, used, ext, contexts, loadavg };
};
```

/api/signIn.js

```
({
  access: 'public',
  method: async ({ login, password }) => {
    const user = await application.auth.getUser(login);
    const hash = user ? user.password : undefined;
    const valid = await application.security.validatePassword(password, hash);
    if (!user || !valid) throw new Error('Incorrect login or password');
```

```
    console.log(`Logged user: ${login}`);
      return { result: 'success', userId: user.id };
    }
  });
```

/api/status.js

```
  async () => {
    const status = context.token ? 'logged' : 'not logged';
    return { result: status };
  };
```

/cert/generate.sh

```
  cd "$(dirname "$0")"
  openssl genrsa -out key.pem 2048
  openssl req -new -out self.pem -key key.pem -subj '/CN=localhost'
  openssl req -text -noout -in self.pem
  openssl x509 -req -days 1024 -in self.pem -signkey key.pem -out cert.pem -
extfile generate.ext
```

/config/database.js

```
  ({
    host: '127.0.0.1',
    port: 27017,
    database: 'application',
    user: 'marcus',
    password: 'marcus',
  });
```

/config/resmon.js

```
({
  interval: 30000,
});
```

/config/server.js

```
({
  host: '127.0.0.1',
  ports: [8000, 8001, 8002, 8003],
  timeout: 5000,
  concurrency: 1000,
  queue: {
    size: 2000,
    timeout: 3000,
  },
});
```

/domain/resmon.js

```
({
  getStatistics() {
    const { heapTotal, heapUsed, external } = api.process.memoryUsage();
    const hs = api.v8.getHeapStatistics();
    const contexts = hs.number_of_native_contexts;
    const detached = hs.number_of_detached_contexts;
    return { heapTotal, heapUsed, external, contexts, detached };
  },
```

```
  start() {
    const { interval } = application.resmon.config;
    setInterval(() => {
      const stats = application.resmon.getStatistics();
      const { heapTotal, heapUsed, external, contexts, detached } = stats;
      const total = application.utils.bytesToSize(heapTotal);
      const used = application.utils.bytesToSize(heapUsed);
      const ext = application.utils.bytesToSize(external);
      console.log(`Heap: ${used} of ${total}, ext: ${ext}`);
      console.log(`Contexts: ${contexts}, detached: ${detached}`);
    }, interval);
  }
});
```

/domain/utils.js

```
({
  bytesToSize(bytes) {
    const UNITS = ['', ' Kb', ' Mb', ' Gb', ' Tb', ' Pb', ' Eb', ' Zb', ' Yb'];
    if (bytes === 0) return '0';
    const exp = Math.floor(Math.log(bytes) / Math.log(1000));
    const size = bytes / 1000 ** exp;
    const short = Math.round(size, 2);
    const unit = UNITS[exp];
    return short + unit;
  }
});
```

/lib/application.js

```
'use strict';

const api = require('./dependencies.js');

const { path, events, vm, fs, fsp } = api;

const security = require('./security.js');

const SCRIPT_OPTIONS = { timeout: 5000 };

const DIRS = ['static', 'domain', 'api'];

class Application extends events.EventEmitter {
  constructor() {
    super();
    this.finalization = false;
    this.namespaces = ['db'];
    this.path = process.cwd();
    this.staticPath = path.join(this.path, 'static');
  }
  async init() {
    for (const name of DIRS) {
      this[name] = new Map();
      await this.loadPlace(name, path.join(this.path, name));
    }
  }
  async shutdown() {
    this.finalization = true;
    await this.server.close();
```

```
}
createSandbox() {
  const introspection = async () => [...this.api.keys()];
  const context = Object.freeze({});
  const application = { security, context, introspection };
  for (const name of this.namespaces) application[name] = this[name];
  const sandbox = {
    console: this.logger, Buffer, application, api,
    setTimeout, setImmediate, setInterval,
    clearTimeout, clearImmediate, clearInterval,
  };
  sandbox.global = sandbox;
  return vm.createContext(sandbox);
}
sandboxInject(name, module) {
  this[name] = Object.freeze(module);
  this.namespaces.push(name);
}
async createScript(fileName) {
  const code = await fsp.readFile(fileName, 'utf8');
  const src = '\'use strict\';\n' + code;
  const options = { filename: fileName, lineOffset: -1 };
  try {
    return new vm.Script(src, options);
  } catch (err) {
```

```
    this.logger.error(err.stack);
    return null;
  }
}
runScript(methodName, session) {
  const { sandbox } = session || this;
  const script = this.api.get(methodName);
  if (!script) return null;
  const exp = script.runInContext(sandbox, SCRIPT_OPTIONS);
  return typeof exp !== 'object' ? { access: 'logged', method: exp } : exp;
}
async loadFile(filePath) {
  const key = filePath.substring(this.staticPath.length);
  try {
    const data = await fsp.readFile(filePath);
    this.static.set(key, data);
  } catch (err) {
    this.logger.error(err.stack);
    if (err.code !== 'ENOENT') throw err;
  }
}
async loadScript(place, fileName) {
  const { name, ext } = path.parse(fileName);
  if (ext !== '.js' || name.startsWith('.')) return;
  const script = await this.createScript(fileName);
```

```
  const map = this[place];
  if (!script) {
    map.delete(name);
    return;
  }
  if (place === 'domain') {
    const config = this.config.sections[name];
    this.sandbox.application[name] = { config };
    const exp = script.runInContext(this.sandbox, SCRIPT_OPTIONS);
    if (config) exp.config = config;
    this.sandbox.application[name] = exp;
    this.sandboxInject(name, exp);
    if (exp.start) exp.start();
  } else {
    map.set(name, script);
  }
}
async loadPlace(place, placePath) {
  const files = await fsp.readdir(placePath, { withFileTypes: true });
  for (const file of files) {
    const filePath = path.join(placePath, file.name);
    if (place !== 'static') await this.loadScript(place, filePath);
    else if (file.isDirectory()) await this.loadPlace(place, filePath);
    else await this.loadFile(filePath);
  }
```

```
    fs.watch(placePath, (event, fileName) => {
      const filePath = path.join(placePath, fileName);
      if (place === 'static') this.loadFile(filePath);
      else this.loadScript(place, filePath);
    });
  }
}
module.exports = new Application();
```

/lib/auth.js

```
'use strict';
const { crypto, common } = require('./dependencies.js');
const application = require('./application.js');
const BYTE  = 256;
const TOKEN = 'token';
const TOKEN_LENGTH = 32;
const ALPHA_UPPER = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ';
const ALPHA_LOWER = 'abcdefghijklmnopqrstuvwxyz';
const ALPHA = ALPHA_UPPER + ALPHA_LOWER;
const DIGIT = '0123456789';
const ALPHA_DIGIT = ALPHA + DIGIT;
const EPOCH = 'Thu, 01 Jan 1970 00:00:00 GMT';
const FUTURE = 'Fri, 01 Jan 2100 00:00:00 GMT';
const LOCATION = 'Path=/; Domain';
const COOKIE_DELETE = `${TOKEN}=deleted; Expires=${EPOCH}; ${LOCATION
```

```
}=`;
const COOKIE_HOST = `Expires=${FUTURE}; ${LOCATION}`;
const SANDBOX_POOL = 20;
const sessions = new Map();
const cache = new WeakMap();
const pool = [];
const generateToken = () => {
  const base = ALPHA_DIGIT.length;
  const bytes = crypto.randomBytes(base);
  let key = '';
  for (let i = 0; i < TOKEN_LENGTH; i++) {
    const index = ((bytes[i] * base) / BYTE) | 0;
    key += ALPHA_DIGIT[index];
  }
  return key;
};
const parseCookies = cookie => {
  const values = {};
  const items = cookie.split(';');
  for (const item of items) {
    const parts = item.split('=');
    const key = parts[0].trim();
    const val = parts[1] || '';
    values[key] = val.trim();
  }
  return values;
```

```
};

module.exports = () => {
  let timer = null;
  const { db } = application;
  const fillPool = () => {
    const need = SANDBOX_POOL - pool.length;
    for (let i = 0; i < need; i++) {
      pool.push(application.createSandbox());
    }
  };
  const getSandbox = () => {
    if (timer === null) {
      timer = setTimeout(() => {
        fillPool();
        timer = null;
      }, 0);
    }
    if (pool.length > 0) return pool.pop();
    return application.createSandbox();
  };
  const save = (token, context) => {
    const data = JSON.stringify(context);
    db.update('Session', { data }, { token });
  };
  class Session {
```

```
  constructor(token, sandbox, contextData = { token }) {
    const contextHandler = {
      set: (data, key, value) => {
        const res = Reflect.set(data, key, value);
        save(token, this.data);
        return res;
      }
    };
    this.token = token;
    this.sandbox = sandbox;
    this.data = contextData;
    sandbox.context = new Proxy(contextData, contextHandler);
  }
}
const start = (client, userId) => {
  const token = generateToken();
  const host = common.parseHost(client.req.headers.host);
  const ip = client.req.connection.remoteAddress;
  const cookie = `${TOKEN}=${token}; ${COOKIE_HOST}=${host}; HttpOnly`;
  const sandbox = getSandbox();
  const session = new Session(token, sandbox);
  sessions.set(token, session);
  cache.set(client.req, session);
  const data = JSON.stringify(session.data);
  db.insert('Session', { userId, token, ip, data });
```

```
  if (client.res) client.res.setHeader('Set-Cookie', cookie);
  return session;
};
const restore = async client => {
  const cachedSession = cache.get(client.req);
  if (cachedSession) return cachedSession;
  const { cookie } = client.req.headers;
  if (!cookie) return null;
  const cookies = parseCookies(cookie);
  const token = cookies.token;
  if (!token) return null;
  let session = sessions.get(token);
  if (!session) {
    const [record] = await db.select('Session', ['Data'], { token });
    if (record && record.data) {
      const data = JSON.parse(record.data);
      const sandbox = getSandbox();
      session = new Session(token, sandbox, data);
      sessions.set(token, session);
    }
  }
  if (!session) return null;
  cache.set(client.req, session);
  return session;
};
```

```
  const remove = (client, token) => {
    const host = common.parseHost(client.req.headers.host);
    client.res.setHeader('Set-Cookie', COOKIE_DELETE + host);
    sessions.delete(token);
    db.delete('Session', { token });
  };
  const registerUser = (login, password, fullName) => {
    db.insert('SystemUser', { login, password, fullName });
  };
  const getUser = login => db
    .select('SystemUser', ['Id', 'Password'], { login })
    .then(([user]) => user);
  return { fillPool, start, restore, remove, save, registerUser, getUser };
};
```

/lib/client.js

```
'use strict';
const { http, path } = require('./dependencies.js');
const application = require('./application.js');
const MIME_TYPES = {
  html: 'text/html; charset=UTF-8',
  json: 'application/json; charset=UTF-8',
  js: 'application/javascript; charset=UTF-8',
  css: 'text/css',
  png: 'image/png',
```

```
  ico: 'image/x-icon',
  svg: 'image/svg+xml',
};
const HEADERS = {
  'X-XSS-Protection': '1; mode=block',
  'X-Content-Type-Options': 'nosniff',
  'Strict-Transport-Security': 'max-age=31536000; includeSubdomains; preload',
  'Access-Control-Allow-Origin': '*',
  'Content-Security-Policy': [
    'default-src \'self\'',
    'style-src \'self\' https://fonts.googleapis.com',
    'font-src \'self\' https://fonts.gstatic.com',
  ].join('; '),
};

class Client {
  constructor(req, res, connection) {
    this.req = req;
    this.res = res;
    this.connection = connection;
  }
  static() {
    const { req: { url }, res } = this;
    const filePath = url === '/' ? '/index.html' : url;
    const fileExt = path.extname(filePath).substring(1);
```

```
    const mimeType = MIME_TYPES[fileExt] || MIME_TYPES.html;
    res.writeHead(200, { ...HEADERS, 'Content-Type': mimeType });
    const data = application.static.get(filePath);
    if (data) res.end(data);
    else this.error(404);
  }
  redirect(location) {
    const { res } = this;
    if (res.headersSent) return;
    res.writeHead(302, { 'Location': location });
    res.end();
  }
  error(status, err) {
    const { req: { url }, res, connection } = this;
    const reason = http.STATUS_CODES[status];
    const error = err ? err.stack : reason;
    const msg = status === 403 ? err.message : `${url}- ${error} - ${status}`;
    application.logger.error(msg);
    const result = JSON.stringify({ result: 'error', reason });
    if (connection) {
      connection.send(result);
      return;
    }
    if (res.finished) return;
    res.writeHead(status, { 'Content-Type': MIME_TYPES.json });
```

```
  res.end(result);
}
async rpc(method, args) {
  const { res, connection } = this;
  const { semaphore } = application.server;
  try {
    await semaphore.enter();
  } catch {
    this.error(504);
    return;
  }
  try {
    const session = await application.auth.restore(this);
    const proc = application.runScript(method, session);
    if (!proc) {
      this.error(404);
      return;
    }
    if (!session && proc.access !== 'public') {
      this.error(403, new Error(`Forbidden: /api/${method}`));
      return;
    }
    const result = await proc.method(args);
    if (!session && proc.access === 'public') {
      const session = application.auth.start(this, result.userId);
```

```
        result.token = session.token;
      }
      const data = JSON.stringify(result);
      if (connection) connection.send(data);
      else res.end(data);
    } catch (err) {
      this.error(500, err);
    } finally {
      semaphore.leave();
    }
  }
}
module.exports = Client;
```

/lib/common.js

```
'use strict';
const parseHost = host => {
  const portOffset = host.indexOf(':');
  if (portOffset > -1) return host.substr(0, portOffset);
  return host;
};
const timeout = msec => new Promise(resolve => {
  setTimeout(resolve, msec);
});
const sample = arr => arr[Math.floor(Math.random() * arr.length)];
```

```
module.exports = { parseHost, timeout, sample };
```

/lib/config.js

```
'use strict';
const { path, fsp, vm } = require('./dependencies.js');
const SCRIPT_OPTIONS = { timeout: 5000 };
class Config {
  constructor(configPath) {
    this.sections = {};
    this.path = configPath;
    this.sandbox = vm.createContext({});
    return this.load();
  }
  async load() {
    const files = await fsp.readdir(this.path);
    for (const fileName of files) {
      await this.loadFile(fileName);
    }
    return this;
  }
  async loadFile(fileName) {
    const { name, ext } = path.parse(fileName);
    if (ext !== '.js') return;
    const configFile = path.join(this.path, fileName);
    const code = await fsp.readFile(configFile, 'utf8');
```

```
    const src = `'use strict';\n${code}`;
    const options = { filename: configFile, lineOffset: -1 };
    const script = new vm.Script(src, options);
    const exports = script.runInContext(this.sandbox, SCRIPT_OPTIONS);
    this.sections[name] = exports;
  }
}
module.exports = Config;
```

/lib/database.js

```
'use strict';
const { Pool } = require('pg');
const application = require('./application.js');
const OPERATORS = ['>=', '<=', '<>', '>', '<'];
const where = (conditions, firstArgIndex = 1) => {
  const clause = [];
  const args = [];
  let i = firstArgIndex;
  const keys = Object.keys(conditions);
  for (const key of keys) {
    let operator = '=';
    let value = conditions[key];
    if (typeof value === 'string') {
      for (const op of OPERATORS) {
        const len = op.length;
```

```
        if (value.startsWith(op)) {
          operator = op;
          value = value.substring(len);
        }
      }
      if (value.includes('*') || value.includes('?')) {
        operator = 'LIKE';
        value = value.replace(/\*/g, '%').replace(/\?/g, '_');
      }
    }
    clause.push(`${key} ${operator} $${i++}`);
    args.push(value);
  }
  return { clause: clause.join(' AND '), args };
};
const updates = (delta, firstArgIndex = 1) => {
  const clause = [];
  const args = [];
  let i = firstArgIndex;
  const keys = Object.keys(delta);
  for (const key of keys) {
    const value = delta[key].toString();
    clause.push(`${key} = $${i++}`);
    args.push(value);
  }
```

```
  return { clause: clause.join(', '), args };
};
class Database {
  constructor(config) {
    this.pool = new Pool(config);
  }
  query(sql, values) {
    const data = values ? values.join(',') : '';
    application.logger.log(`${sql}\t[${data}]`);
    return this.pool.query(sql, values);
  }
  insert(table, record) {
    const keys = Object.keys(record);
    const nums = new Array(keys.length);
    const data = new Array(keys.length);
    let i = 0;
    for (const key of keys) {
      data[i] = record[key];
      nums[i] = `$${++i}`;
    }
    const fields = keys.join(', ');
    const params = nums.join(', ');
    const sql = `INSERT INTO ${table} (${fields}) VALUES (${params})`;
    return this.query(sql, data);
  }
```

```
async select(table, fields = ['*'], conditions = null) {
  const keys = fields.join(', ');
  const sql = `SELECT ${keys} FROM ${table}`;
  let whereClause = '';
  let args = [];
  if (conditions) {
    const whereData = where(conditions);
    whereClause = ' WHERE ' + whereData.clause;
    args = whereData.args;
  }
  const res = await this.query(sql + whereClause, args);
  return res.rows;
}
delete(table, conditions = null) {
  const { clause, args } = where(conditions);
  const sql = `DELETE FROM ${table} WHERE ${clause}`;
  return this.query(sql, args);
}
update(table, delta = null, conditions = null) {
  const upd = updates(delta);
  const cond = where(conditions, upd.args.length + 1);
  const sql = `UPDATE ${table} SET ${upd.clause} WHERE ${cond.clause}`;
  const args = [...upd.args, ...cond.args];
  return this.query(sql, args);
}
```

```
  close() {
    this.pool.end();
  }
}
module.exports = Database;
```

/lib/dependencies.js

```
'use strict';
const api = { common: require('./common.js') };
const internals = [
  'util', 'child_process', 'worker_threads', 'os', 'v8', 'vm', 'path', 'url',
  'assert', 'querystring', 'string_decoder', 'perf_hooks', 'async_hooks',
  'timers', 'events', 'stream', 'fs', 'crypto', 'zlib',
  'dns', 'net', 'tls', 'http', 'https', 'http2', 'dgram',
];
for (const name of internals) api[name] = Object.freeze(require(name));
api.process = process;
api.childProcess = api['child_process'];
api.StringDecoder = api['string_decoder'];
api.perfHooks = api['perf_hooks'];
api.asyncHooks = api['async_hooks'];
api.worker = api['worker_threads'];
api.fsp = api.fs.promises;
module.exports = Object.freeze(api);
```

/lib/logger.js

```
'use strict';

const { fs, util, path } = require('./dependencies.js');

const COLORS = {
  info: '\x1b[1;37m',
  debug: '\x1b[1;33m',
  error: '\x1b[0;31m',
};

class Logger {
  constructor(logPath, threadId = 0) {
    this.path = logPath;
    const date = new Date().toISOString().substring(0, 10);
    const filePath = path.join(logPath, `${date}-W${threadId}.log`);
    this.stream = fs.createWriteStream(filePath, { flags: 'a' });
    this.regexp = new RegExp(path.dirname(this.path), 'g');
  }
  write(level = 'info', s) {
    const date = new Date().toISOString();
    const color = COLORS[level];
    const line = date + '\t' + s;
    console.log(color + line + '\x1b[0m');
    const out = line.replace(/[\n\r]\s*/g, '; ') + '\n';
    this.stream.write(out);
  }
```

```
  log(...args) {
    const msg = util.format(...args);
    this.write('info', msg);
  }
  dir(...args) {
    const msg = util.inspect(...args);
    this.write('info', msg);
  }
  debug(...args) {
    const msg = util.format(...args);
    this.write('debug', msg);
  }
  error(...args) {
    const msg = util.format(...args).replace(/[\n\r]{2,}/g, '\n');
    this.write('error', msg.replace(this.regexp, ''));
  }
}
module.exports = Logger;
```

/lib/security.js

```
'use strict';
const { crypto } = require('./dependencies.js');
const serializeHash = (hash, salt, params) => {
  const paramString = Object.entries(params)
    .map(([key, value]) => `${key}=${value}`).join(',');
```

```
  const saltString = salt.toString('base64').split('=')[0];
  const hashString = hash.toString('base64').split('=')[0];
  return `$scrypt$${paramString}$${saltString}$${hashString}`;
};
const deserializeHash = phcString => {
  const parsed = phcString.split('$');
  parsed.shift();
  if (parsed[0] !== 'scrypt') {
    throw new Error('Node.js crypto module only supports scrypt');
  }
  const params = Object.fromEntries(
    parsed[1].split(',').map(p => {
      const kv = p.split('=');
      kv[1] = Number(kv[1]);
      return kv;
    })
  );
  const salt = Buffer.from(parsed[2], 'base64');
  const hash = Buffer.from(parsed[3], 'base64');
  return { params, salt, hash };
};

const SALT_LEN = 32;
const KEY_LEN = 64;
// Only change these if you know what you're doing
```

```
const SCRYPT_PARAMS = {
  N: 32768,
  r: 8,
  p: 1,
  maxmem: 64 * 1024 * 1024,
};
const hashPassword = password => new Promise((resolve, reject) => {
  crypto.randomBytes(SALT_LEN, (err, salt) => {
    if (err) {
      reject(err);
      return;
    }
    crypto.scrypt(password, salt, KEY_LEN, SCRYPT_PARAMS, (err, hash) => {
      if (err) {
        reject(err);
        return;
      }
      resolve(serializeHash(hash, salt, SCRYPT_PARAMS));
    });
  });
});
let defaultHash;
hashPassword('').then(hash => {
  defaultHash = hash;
});
```

```
const validatePassword = (password, hash = defaultHash) =>
  new Promise((resolve, reject) => {
    const parsedHash = deserializeHash(hash);
    const len = parsedHash.hash.length;
    crypto.scrypt(password, parsedHash.salt, len, parsedHash.params,
      (err, hashedPassword) => {
        if (err) {
          reject(err);
          return;
        }
        resolve(crypto.timingSafeEqual(hashedPassword, parsedHash.hash));
      }
    );
  });
module.exports = Object.freeze({ hashPassword, validatePassword });
```

/lib/semaphore.js

```
'use strict';
class Semaphore {
  constructor(concurrency, size, timeout) {
    this.counter = concurrency;
    this.timeout = timeout;
    this.size = size;
    this.queue = [];
  }
```

```
enter() {
  return new Promise((resolve, reject) => {
    if (this.counter > 0) {
      this.counter--;
      resolve();
      return;
    }
    if (this.queue.length >= this.size) {
      reject(new Error('Semaphore queue is full'));
      return;
    }
    const timer = setTimeout(() => {
      reject(new Error('Semaphore timeout'));
    }, this.timeout);
    this.queue.push({ resolve, timer });
  });
}
leave() {
  if (this.queue.length === 0) {
    this.counter++;
    return;
  }
  const { resolve, timer } = this.queue.shift();
  clearTimeout(timer);
  setTimeout(() => {
```

```
      resolve();
    }, 0);
  }
}
module.exports = Semaphore;
```

/lib/server.js

```
'use strict';
const { http, https, worker, common } = require('./dependencies.js');
const application = require('./application.js');

const WebSocket = require('ws');
const Semaphore = require('./semaphore.js');
const Client = require('./client.js');
const SHUTDOWN_TIMEOUT = 5000;
const LONG_RESPONSE = 30000;
const METHOD_OFFSET = '/api/'.length;
const clients = new Map();
const receiveArgs = async req => new Promise(resolve => {
  const body = [];
  req.on('data', chunk => {
    body.push(chunk);
  }).on('end', async () => {
    const data = body.join('');
    const args = JSON.parse(data);
```

```
    resolve(args);
  });
});
const closeClients = () => {
  for (const [connection, client] of clients.entries()) {
    clients.delete(connection);
    client.error(503);
    connection.destroy();
  }
};
const listener = (req, res) => {
  let finished = false;
  const { method, url, connection } = req;
  const client = new Client(req, res);
  clients.set(connection, client);
  const timer = setTimeout(() => {
    if (finished) return;
    finished = true;
    clients.delete(connection);
    client.error(504);
  }, LONG_RESPONSE);
  res.on('close', () => {
    if (finished) return;
    finished = true;
    clearTimeout(timer);
```

```
    clients.delete(connection);
  });
  application.logger.log(`${method}\t${url}`);
  if (url.startsWith('/api/')) {
    if (method !== 'POST') {
      client.error(403, new Error(`Forbidden: ${url}`));
      return;
    }
    receiveArgs(req).then(args => {
      const method = url.substring(METHOD_OFFSET);
      client.rpc(method, args);
    });
  } else {
    if (url === '/' && !req.connection.encrypted) {
      const host = common.parseHost(req.headers.host);
      const port = common.sample(application.server.ports);
      client.redirect(`https://${host}:${port}/`);
    }
    client.static();
  }
};
class Server {
  constructor(config) {
    this.config = config;
    const { ports, host, concurrency, queue } = config;
```

```
    this.semaphore = new Semaphore(concurrency, queue.size, queue.timeout);
    const { threadId } = worker;
    const port = ports[threadId - 1];
    this.ports = config.ports.slice(1);
    const transport = threadId === 1 ? http : https;
    this.instance = transport.createServer({ ...application.cert }, listener);
    this.ws = new WebSocket.Server({ server: this.instance });
    this.ws.on('connection', (connection, req) => {
      const client = new Client(req, null, connection);
      connection.on('message', message => {
        const { method, args } = JSON.parse(message);
        client.rpc(method, args);
      });
    });
    this.instance.listen(port, host);
  }
  async close() {
    this.instance.close(err => {
      if (err) application.logger.error(err.stack);
    });
    await common.timeout(SHUTDOWN_TIMEOUT);
    closeClients();
  }
}
module.exports = Server;
```

/lib/worker.js

```
'use strict';

const { worker, fsp, path } = require('./dependencies.js');

const application = require('./application.js');

const Config = require('./config.js');

const Logger = require('./logger.js');

const Database = require('./database.js');

const Server = require('./server.js');

const initAuth = require('./auth.js');

(async () => {
  const configPath = path.join(application.path, 'config');
  const config = await new Config(configPath);
  const logPath = path.join(application.path, 'log');
  const logger = await new Logger(logPath, worker.threadId);
  const certPath = path.join(application.path, 'cert');
  Object.assign(application, { config, logger });
  try {
    const key = await fsp.readFile(path.join(certPath, 'key.pem'));
    const cert = await fsp.readFile(path.join(certPath, 'cert.pem'));
    application.cert = { key, cert };
  } catch {
    if (worker.threadId === 1) logger.log('Can not load TLS certificates');
  }
  application.db = new Database(config.sections.database);
```

```
  application.server = new Server(config.sections.server);
  application.auth = initAuth();
  application.sandboxInject('auth', application.auth);
  application.sandbox = application.createSandbox();
  await application.init();
  application.auth.fillPool();
  logger.log(`Application started in worker ${worker.threadId}`);
  worker.parentPort.on('message', async message => {
    if (message.name === 'stop') {
      if (application.finalization) return;
      logger.log(`Graceful shutdown in worker ${worker.threadId}`);
      await application.shutdown();
      process.exit(0);
    }
  });
  const logError = err => {
    logger.error(err.stack);
  };
  process.on('uncaughtException', logError);
  process.on('warning', logError);
  process.on('unhandledRejection', logError);
})();

/server.js
'use strict';
```

```
const { Worker } = require('worker_threads');
const path = require('path');
const Config = require('./lib/config.js');
const PATH = process.cwd();
const CFG_PATH = path.join(PATH, 'config');
(async () => {
  const config = await new Config(CFG_PATH);
  const { sections } = config;
  const count = sections.server.ports.length;
  const workers = new Array(count);
  const start = id => {
    const worker = new Worker('./lib/worker.js');
    workers[id] = worker;
    worker.on('exit', code => {
      if (code !== 0) start(id);
    });
  };
  for (let id = 0; id < count; id++) start(id);
  const stop = async () => {
    console.log();
    for (const worker of workers) {
      worker.postMessage({ name: 'stop' });
    }
  };
  process.on('SIGINT', stop);
```

```
  process.on('SIGTERM', stop);
})();
```

./package.json

```
{
  "name": "servermetrics",
  "version": "1.0.0",
  "private": true,
  "description": "System for collecting server metric data ",
  "author": "Vitaliy Pavlenko",
  "license": "MIT",
  "keywords": [
    "server",
    "api",
    "rpc",
    "rest",
    "service",
    "web",
    "router",
    "routing",
    "cluster",
    "cache",
    "http",
    "https",
    "websocket",
```

```
  "websockets",
  "samdboxing",
  "isolation",
  "context",
  "framework",
  "scale",
  "scaling",
  "thread",
  "worker"
],
"main": "server.js",
"scripts": {
  "test": "npm run -s lint && node test/all.js",
  "lint": "eslint .",
  "install": "cert/generate.sh"
},
"engines": {
  "node": ">=12.5.0"
},
"devDependencies": {
  "eslint": "^7.0.0"
},
"dependencies": {
  "pg": "^8.1.1",
  "ws": "^7.3.0"
```

```
    }

}
```